F3

Fakulta elektrotechnická
Katedra počítačů

Diplomová práce

Softwarový modul pro podporu telerehabilitace v projektu TERESA

Bc. Vojtěch Novotný

Květen 2023
Vedoucí práce: doc. Ing. Miroslav Bureš, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Novotný** Jméno: **Vojtěch** Osobní číslo: **466339**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Softwarový modul pro podporu telerehabilitace v projektu Teresa

Název diplomové práce anglicky:

Software module for telerehablitation support in the Teresa project

Pokyny pro vypracování:

Navrhněte a implementujte softwarový modul, který bude umožňovat zobrazení sady cvičení rehabilitovaným pacientům v mobilní aplikaci. Konkrétní cviky v rámci rehabilitačního plánu budou individuálně přiřazovány jednotlivým pacientům lékařem nebo fyzioterapeutem pomocí administračního rozhraní implementovaného jako webová aplikace. V tomto rozhraní bude systém umožňovat vytvářet nové cviky a spravovat je pomocí katalogu. Jako formát bude systém umožňovat textový popis cviku kombinovaný se sadou fotografií, případně videozáznamem cviku. Kromě toho budou mít pacienti v mobilní aplikaci k dispozici interaktivní dotazníky, kterými budou komunikovat s odborným personálem. Systém otestujte sadou vhodných uživatelských testů.

Seznam doporučené literatury:

Bures, M., Neumannova, K., Blazek, P., Klima, M., Schvach, H., Nema, J., ... & Koblizek, V. (2022). A Sensor Network Utilizing Consumer Wearables for Telerehabilitation of Post-Acute COVID-19 Patients. IEEE Internet of Things Journal, 9(23), 23795-23809.
Peretti, A., Amenta, F., Tayebati, S. K., Nittari, G., & Mahdi, S. S. (2017). Telerehabilitation: review of the state-of-the-art and areas of application. JMIR rehabilitation and assistive technologies, 4(2), e7511.
Hailey, D., Roine, R., Ohinmaa, A., & Dennett, L. (2011). Evidence of benefit from telerehabilitation in routine care: a systematic review. Journal of telemedicine and telecare, 17(6), 281-287.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Miroslav Bureš, Ph.D. laboratoř inteligentního testování systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **08.02.2023** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **22.09.2024**

doc. Ing. Miroslav Bureš, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání Podpis studenta

Poděkování / Prohlášení

Chtěl bych poděkoval svému vedoucímu práce doc. Ing. Miroslavu Burešovi, Ph.D. za podporu během vypracovávání diplomové práce. Dále bych rád poděkoval doc. Kateřině Neumannové, Ph.D. za pomoc při sběru a ověřování požadavků pro modul rehabilitačních cvičení v systému TERESA.

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.

V Praze dne 18. 5. 2023

.....

Abstrakt / Abstract

Tato práce se zabývá návrhem a implementací softwarového modulu pro správu rehabilitačních plánů v rámci systému projektu TERESA (TEleREhabilitation Self-training Assistant). Výstupem práce je administrační webové rozhraní pro lékaře poskytující správu katalogu cviků a rehabilitačních plánů a multiplatformní mobilní aplikace pro pacienty zobrazující jim přidělené cviky. Modul poskytuje získávání zpětné vazby od pacientů pomocí dotazníků v mobilní aplikaci.

Práce zahrnuje sběr a analýzu požadavků, návrh architektury a uživatelského rozhraní systému a samotnou implementaci navrženého řešení. V závěru práce je systém otestován automatickými end-to-end testy.

Klíčová slova: telerehabilitace; telemedicína; mobilní aplikace; TERESA; Spring; React Native.

This thesis deals with the design and implementation of a software module for managing rehabilitation plans within the TERESA (TEleREhabilitation Self-training Assistant) system. The output of the thesis is an administrative web interface for doctors providing management of the catalog of exercises and rehabilitation plans and a multi-platform mobile application for patients displaying the exercises assigned to them. The module provides for obtaining feedback from patients using questionnaires in the mobile application.

The work includes the collection and analysis of requirements, the design of the architecture and user interface of the system, and the actual implementation of the proposed solution. At the end of the work, the system is tested with automated end-to-end tests.

Keywords: telemedicine; telerehabilitation; mobile health; TERESA; Spring; React Native.

Title translation: Software module for telerehablitation support in the TERESA project

Obsah /

/ Obrázky

Kapitola 1
Úvod

1.1 Stávající systém projektu TERESA

V roce 2021 začaly čtyři týmy odborníků z Českého vysokého učení technického v Praze (ČVUT), Fakultní nemocnice Hradec Králové (FN HK), Univerzity Palackého v Olomouci (UPOL) a Univerzity obrany (UNOB) spolupracovat na projektu TERESA (TEleREhabilitation Self-training Assistant) [1]. Jedná se o projekt, jehož původním cílem bylo vytvořit systém pro telerehabilitaci pacientů s přetrvávajícími následky po prodělaném onemocnění COVID-19 a dalšími plicními onemocněními.

Systém byl založený na sběru dat o pohybové aktivitě pacientů ze snadno dostupných fitness náramků [2]. Tato data byla přenášena na server, kde byla zpracována do přehledných denních reportů pro fyzioterapeuty. Tyto reporty poté sloužily jako vhodný podklad pro sestavení a vedení rehabilitačních plánů pro sledované pacienty.

Stávající systém se sestává z chytrých náramků, mobilní aplikace a webového rozhraní pro lékaře a fyzioterapeuty. Naměřená data z chytrých náramků jsou aplikací pravidelně odesílána na webový server, kde jsou ukládána v databázi. Webová aplikace poté autentizovaným uživatelům poskytuje přehled naměřených dat pacientů a možnost z těchto přehledů generovat týdenní reporty. Webová aplikace také posyktuje správu denních dotazníků o průběhu cvičení, které se pacientům zobrazují v mobilní aplikaci.

Ačkoliv systém poskytuje cenný sběr dat pro vytváření rehabilitačních plánů, samotné rehabilitace poté probíhají standardním způsobem, tedy pravidelnými návštěvami pacienta v nemocnici. Z tohoto důvodu se nabízí rozšíření současného systému o správu cviků a rehabilitačních plánů, ke kterým bude mít pacient přístup ze svého mobilního zařízení. Digitalizace celého procesu rehabilitace tak přinese jak finanční tak časovou úsporu lékařům i jejich pacientům.

1.2 Telerehabilitace

Telerehabilitace je jedním z druhů zdravotní péče poskytující rehabilitační služby pomocí telekomunikačních technologií distanční formou. Telerehabilitace byly navrženy pro zvýšení dostupnosti a zlepšení kontinuity péče o zranitelné skupiny obyvatel se zdravotním postižením s potenciální úsporou času a nákladů [3].

Telerehabilitace umožňují pacientům získat rehabilitační služby z pohodlí jejich domova nebo místních klinik bez nutnosti častých osobních návštěv nemocnice nebo rehabilitačního centra. K tomu se využívají různé komunikační technologie jako jsou videokonference, mobilní aplikace, vzdálená monitorovací zařízení a online platformy, které usnadňují interakci mezi pacienty a zdravotnickými pracovníky v reálném čase [4].

Mezi důvody toho, proč se pacienti nezapojují do rehabilitačních programů pod dohledem, patří nedostatek času v jejich rozvrhu, potíže s dosažením určité úrovně v rámci cvičení nebo neochota provádět cvičení ve skupině [5]. V tomto případě se telerehabilitace nabízí jako vhodné řešní díky flexibilitě toho, kde a kdy pacient může cvičit.

Další výhodou telerehabilitace je kontinuita průběžného sledování a podpory pacientů i po jejich propuštění z nemocnic nebo klinik. Využití technologií usnadňuje pravidelnou komunikaci mezi poskytovateli zdravotní péče a pacienty, zajišťuje konzistentní sledování pokroku a včasné úpravy rehabilitačních plánů.

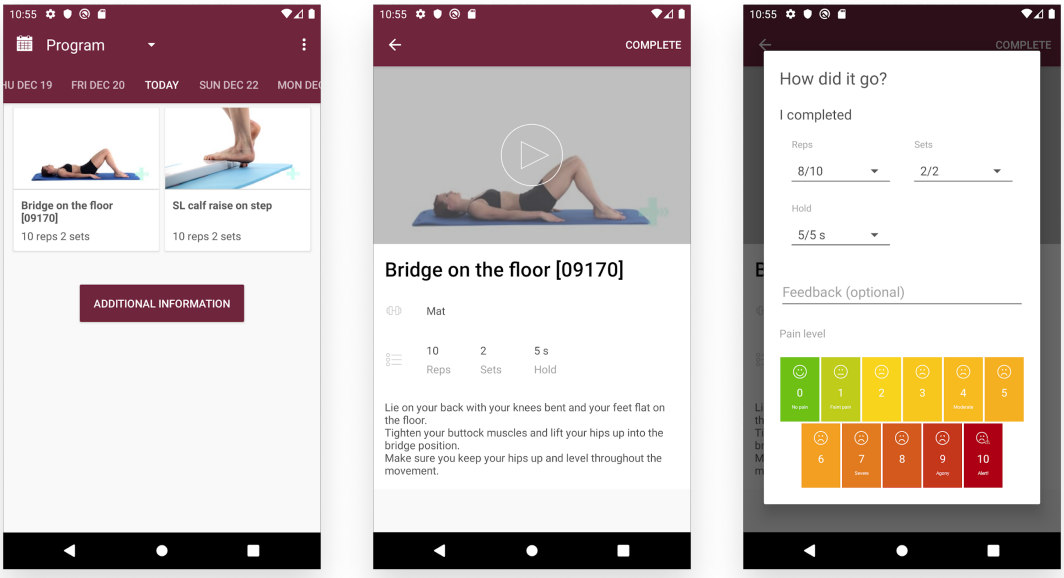
Telerehabilitace také často zahrnují interaktivní a herní prvky (včetně např. virtuální reality), aby byla terapeutická sezení poutavá a zábavná. To může zvýšit motivaci pacienta, což vede k lepšímu dodržování rehabilitačního plánu a zlepšení celkových výsledků [6].

Telerehabilitace se v současnosti využívají zejména ve fyzioterapii ale také při rehabilitacích pacientů s nervovými a srdečními poruchami [5]. Telerehabilitace má však také svá omezení. Nemusí být vhodná pro všechny pacienty, zejména pro ty, kteří mají složité zdravotní problémy vyžadující praktické posouzení nebo intervenci. Problémy mohou představovat technické potíže, jako je špatné připojení k internetu nebo nedostatečná technologická gramotnost. Absence fyzické přítomnosti během terapeutických sezení může omezit schopnost poskytovat praktickou pomoc nebo přímo manipulovat s vybavením.

Přesto, že jsou telerehabilitace v mnoha oblastech zdravotní péče slibným řešením, studie ukazují, že přesvědčivé důkazy o přínosu a dopadu na rutinní rehabilitační programy jsou stále omezené [7] [8]. Pro vyhodnocení účinnosti tohoto přístupu léčby je třeba více detailnějších studií zaměřených především na použití v rutinní léčbě, ne pouze v pilotních projektech.

1.3 Existující řešení

Mezi nejčastější platformy podporující telerehabilitaci patří webové či mobilní aplikace [4]. V současné době existuje celá řada mobilních aplikací poskytující kolekce cviků ve formě instruktážních videí, obrázků a textů.



Obrázek 1.1. Ukázka rehabilitačního plánu v aplikaci Professional Physical Therapy.

Dostupné aplikace na Google Play a App Store se zaměřují především na fyzioterapeutická cvičení (protahování a posilování zad, horních a dolních končetin a další pohybové cviky).

Některé komplexnější aplikace navíc poskytují interakci s lékařem či fyzioterapeutem pomocí chatu nebo videohovoru (např. aplikace HealthQ Rehab Physiotherapy[1]). Mezi další pokročilé funkce patří vytváření rehabilitačních programů a záznam pokroku v těchto programech včetně uvedení obtížnosti u jednotlivých cvičení (viz náhled aplikace Professional Physical Therapy[2] na obrázku 1.1).

Ačkoliv dostupné aplikace umožňují převést většinu rehabilitačního procesu do online prostoru, stále na trhu chybí systém doplňující zmíněné funkce o monitorování pacienta pomocí senzorových sítí a wearables, což je již dostupné v současném systému projektu TERESA (viz sekce 1.1).

1.4 Cíl práce

Cílem práce je rozšířit stávající systém projektu TERESA o modul poskytující telerehabilitace. To bude zahrnovat vytvoření nových funkcí ve webovém rozhraní pro lékaře, ve kterém kromě přehledů dat z fitness náramků pacientů přibydou podstránky pro nahrávání a správu cviků a vytváření rehabilitačních plánů z těchto cviků. Lékař či fyzioterapeut tak bude mít přehled o fyzické aktivitě pacienta, na základě kterého bude moci vytvářet a upravovat jeho rehabilitační plán na míru.

Pacient bude rehabilitačním plánem provázen pomocí mobilní aplikace, která bude propojená s webovým rozhraním lékaře. Tato aplikace bude podobně jako u zmíněných telerehabilitačních aplikací poskytovat možnost označit, která cvičení pacient již dokončil a zda při cvičení měl nějaké obtíže. Tyto informace poté budou viditelné v systému lékaře.

Práce popíše všechny fáze vývoje tohoto modulu od sběru konkrétních požadavků na systém po jeho implmementaci a testování.

[1] `https://play.google.com/store/apps/details?id=com.healthqrehab.app`
[2] `https://play.google.com/store/apps/details?id=com.physitrack.physiapp.propt`

Kapitola 2
Sběr a analýza požadavků

Tato kapitola se věnuje specifikaci požadavků na vyvíjený systém. Nejprve jsou popsány jednotlivé fáze v procesu analýzy požadavků (tzv. Requirements Engineering) a následně je zde uvedena výsledná specifikace funkčních i nefunkčních požadavků na webovou i mobilní část systému (sekce 2.2). Nakonec jsou ze získaných požadavků vytvořeny případy užití pro koncové uživatele systému (sekce 2.3) a celkový proces použití systému je ilustrován v diagramu aktivit 2.3.3.

2.1 Requirements Engineering

Requirements Engineering je proces získávání, analýzy, dokumentace, validace a správy požadavků na software [9]. Je to kritický aspekt vývoje softwaru, protože požadavky definují funkčnost, výkon a kvalitu výsledného systému.

Requirements Engineering zahrnuje několik fází, z nichž každá hraje důležitou roli v procesu vývoje kvalitního softwaru. Níže jsou popsány hlavní fáze:

- **Získávání požadavků**
  V této fázi pracují analytici se stakeholdery (zainteresovanými stranami) na získávání informací o požadavcích na softwarovou aplikaci. To může být provedeno různými metodami jako jsou rozhovory, průzkumy, workshopy nebo pozorování. Cílem této fáze je identifikovat a zdokumentovat potřeby uživatelů, podnikové cíle a technická omezení, které musí softwarová aplikace splňovat.
- **Analýza požadavků**
  Jakmile jsou požadavky získány, softwaroví inženýři začnou fázi analýzy. Ta zahrnuje přezkoumání a zdokonalení požadavků, aby byly úplné, konzistentní a dosažitelné. Fáze analýzy také zahrnuje prioritizaci požadavků na základě jejich důležitosti a dopadu na softwarovou aplikaci. Mezi techniky analýzy patří také profilování uživatelů pomocí případů užití nebo modelování uživatelských procesů.
- **Specifikace požadavků**
  Dalším krokem je zpracování požadavků do strukturovaného dokumentu, který bude jednotlivé požadavky jasně a stručně specifikovat. Dokument by měl obsahovat katalog jak funkčních tak nefunkčních požadavků. Funkční požadavky popisují, jaké služby by měl systém uživatelům poskytnout. Nefunkční požadavky definují kritéria na kvalitu systému (např. výkon, bezpečnost, dostupnost nebo použitelnost).
- **Validace požadavků**
  V této fázi analytici ověřují zdokumentované požadavky se stakeholdery, aby zajistili, že přesně odrážejí potřeby koncových uživatelů. Ověřování lze provést různými metodami jako jsou workshopy s představením požadavků, odesílání požadavků k připomínkování stakeholdery nebo vytváření prototypů k uživatelskému testování.
- **Správa požadavků**
  Jakmile jsou požadavky ověřeny, musí být řízeny během celého procesu vývoje softwaru. To zahrnuje sledování změn požadavků, sdílení aktualizací se stakeholdery

a zajištění toho, že vyvíjený software stále odpovídá specifikaci. Důležitou součástí údržby je také zaznamenávání vztahů mezi jednotlivými požadavky a zaznamenávání závislostí vznikajících specifikací během procesu vývoje.

2.2 Specifikace požadavků

V rámci práce probíhala pravidelná setkání s doc. Kateřinou Neumannovou z Fakulty tělesné kultury UPOL, během kterých byly zjišťovány a ověřovány funkční i nefunkční požadavky na celý systém. Jednotlivé požadavky byly systematicky dokumentovány a pomocí prototypů uživatelských rozhraní zpětně ověřovány s doc. Neumannovou a dalšími stakeholdery. Celý vývoj probíhal v iterativním stylu, který byl založen na pravidelných týdenních schůzkách. Proces vývoje víceméně odrážel metodiku *Agile*, která je charakteristická inkrementálním přístupem k vývoji softwaru a blízkou spoluprácí se zákazníkem po celou dobu projektu [10].

Sběr a analýza požadavků byly potřebné k návrhu datového modelu systému, k definování procesů, které budou v systému probíhat a ke správnému návrhu uživatelského rozhraní. Jednotlivé požadavky na webovou a mobilní aplikaci jsou pospány v následujících sekcích.

2.2.1 Popis webového rozhraní

Do již existujícího webového rozhraní systému TERESA měly přibýt dva další moduly.

První modul měl poskytovat správu cviků a jejich organizaci pomocí katalogů. Představa zákazníka byla taková, že lékař či skupina lékařů bude mít přístup k vlastnímu účtu ve webové aplikaci, ve kterém bude vytvářet jednotlivé cviky použitelné pro rehabilitační plány buď nahráním instruktážního videa nebo přepoužitím již existujícího cviku z jiného účtu v aplikaci. Z tohoto požadavku vyplývala potřeba vytvoření sdílené databáze cviků napříč všemi účty v aplikaci. Dále bylo požadováno, aby ke každému cvičení mohl lékař přidat instrukce v podobě krátkého textového popisu. Pro sandné vyhledávání měly být cviky organizovány do kategorií podle druhu zaměření (např. trénink dýchacích svalů, protahovací cvičení, balanční cvičení atd.). V případě smazání cviku z jednoho účtu v aplikaci mělo být jeho instruktážní video zachováno, aby bylo stále viditelné a přepoužitelné i pro další účty.

V druhém modulu aplikace měl mít lékař možnost vytvářet a spravovat rehabilitační plány pacientů. Proces rehabilitace by začal schůzkou pacienta s lékařem, při které by si pacient nainstaloval mobilní aplikaci a lékař by ji registroval do systému pomocí vygenerovaného registračního kódu. Webové rozhraní by lékaři zobrazovalo seznam všech registrovaných mobilních aplikací v systému včetně informací o datumu instalace, poslední aktivity pacienta a průběhu aktuální rehabilitace. Lékař by poté mohl vybrat konkrétní aplikaci pacienta a vytvořit v ní nový rehabilitační plán. V rámci rehabilitačního plánu by byly pacientovi přidělovány na konkrétní datum a čas cviky z prvního modulu systému. Lékař by mohl v rehabilitačním plánu nastavit připomínku s určitým předstihem před cvičením, která by pacientovi přišla v podobě notifikace v mobilní aplikaci. Dále by detail plánu lékaři zobrazoval informace o splnění cviků pacientem a jeho zpětnou vazbu z dotazníků.

Na konci každého cviku by měl pacient v dotazníku indikovat, zda měl při cvičení nějaké obtíže. Webové rozhraní mělo obsahovat i podstránku pro oragnizaci těchto odpovědí z dotazníků.

V následujícíh sekcích jsou uvedeny všechny zdokumentované funkční i nefunkční požadavky na webové rozhraní.

2.2.2 Funkční požadavky webového rozhraní

Níže jsou uvedeny všechny funčkní požadavky na oba moduly webového rozhraní. Každý požadavek má své číslo, název, stručný popis požadované funkčnosti a uvedenou míru nutnosti.

- **WA FR 01 - Kategorie cvičení**
  Popis: Systém umožní lékaři vytvořit a smazat kategorii cvičení a změnit její název. Systém bude lékaři poskytovat jak přehled všech vytvořených kategorií tak detail konkrétní kategorie se seznamem cviků, které do ní patří.
  Priorita: Vysoká
- **WA FR 02 - Cvičení**
  Popis: Systém umožní lékaři nahrát, smazat a editovat cvičení. Cvičení bude obsahovat tyto položky: název cvičení, popis cvičení a instruktážní video. Popis cvičení bude upravitelný v "rich text" editoru v detailu cvičení. Instruktážní videa budou nahrávána ve formátu MP4.
  Priorita: Vysoká
- **WA FR 03 - Přesunutí cvičení**
  Popis: Systém umožní lékaři přesunout cvičení do jiné kategorie.
  Priorita: Vysoká
- **WA FR 04 - Databáze videí**
  Popis: Systém bude ukládat všechna nahraná videa do společné databáze. Systém umožní lékaři procházet tuto databázi a přepoužít vybrané video pro nové cvičení.
  Priorita: Střední
- **WA FR 05 - Zachování cvičení při smazání**
  Popis: Systém při smazání kategorie nebo samotného cvičení skryje smazaná cvičení, ale nahraná instruktážní videa a popisy těchto cvičení zůstanou ve společné databázi dostupná pro další lékaře.
  Priorita: Střední
- **WA FR 06 - Registrace mobilní aplikace**
  Popis: Systém umožní lékaři registrovat mobilní aplikace pacientů v rehabilitačním systému pomocí vygenerovaného registračního kódu. Systém poskytne lékaři přehled všech spárovaných mobilních aplikací včetně následujících informací: identifikační kód aplikace, datum spárování a datum a čas poslední aktivity pacienta. Systém umožní smazat mobilní aplikaci ze systému.
  Priorita: Vysoká
- **WA FR 07 - Rehabilitační plán**
  Popis: Systém umožní lékaři vytvořit, smazat nebo přejmenovat rehabilitační plán pro určitou mobilní aplikaci pacienta.
  Priorita: Vysoká
- **WA FR 08 - Cvičení v rehabilitačním plánu**
  Popis: Systém umožní přidávat, mazat a upravovat jednotlivá cvičení v rámci rehabilitačního plánu. Cvičení v plánu budou obsahovat tyto položky:
  - datum a čas cvičení (hodiny a minuty)
  - čas notifikace (15 min, 30 min, 1 hod, 12 hod, 1 den předem)
  - název vybraného cvičení - systém poskytne výběr cvičení na základě zvolené kategorie

  Cvičení budou v plánu seřazena dle data. Systém umožní zobrazit detail cvičení včetně videa a popisu vybraného cvičení.
  Priorita: Vysoká

- **WA FR 09 - Přidání více cvičení najednou**

  Popis: Systém bude lékaři umožňovat přidat více cvičení stejného druhu najednou na základě vybraného časového intervalu. Pro každý den v daném intervalu se vytvoří cvičení ve stejném čase.

  Priorita: Nízká
- **WA FR 10 - Ukončení aktivního rehabilitačního plánu**

  Popis: Systém bude lékaři umožňovat ukončit probíhající rehabilitační plán. Ukončením se plán nesmaže, stane se pouze neaktivním a přesune se do historie rehabilitací. Systém umožní aktivovat vybraný plán z historie. Pro každou mobilní aplikaci pacienta může být aktivní nanejvýš jeden rehabilitační plán.

  Priorita: Vysoká
- **WA FR 11 - Status cvičení v plánu**

  Popis: Systém bude lékaři zobrazovat status jednotlivých cvičení v rehabilitačním plánu. Jednotlivé statusy jsou:
  - Cvičení ještě nedokončeno
  - Cvičení přeskočeno
  - Cvičení dokončeno bez obtíží
  - Cvičení dokončeno s obtížemi

  Priorita: Vysoká
- **WA FR 12 - Dotazník po cvičení**

  Popis: Systém umožní lékaři přidávat, mazat a řadit možné odpovědi na dotazník po skončení cvičení. Dotazník bude pro všechna cvičení stejný.

  Priorita: Vysoká
- **WA FR 13 - Odpověďz dotazníku**

  Popis: Systém zobrazí u každého dokončeného kroku v rehabilitačním plánu pacientovu odpověď na dotazník po skončení cvičení.

  Priorita: Vysoká
- **WA FR 14 - Historie rehabilitací**

  Popis: Systém zobrazí lékaři všechny dokončené rehabilitační plány pro vybranou instanci mobilní aplikace. Pro každý plán systém zobrazí datum začátku a ukončení, poslední aktivitu pacienta a přehled průběhu rehabilitace (statusy jednotlivých cvičení).

  Priorita: Střední
- **WA FR 15 - Smazání rehabilitačního plánu**

  Popis: Systém umožní lékaři kompletně smazat celý rehabilitační plán včetně údajů o dokončení jednotlivých cvičení a odpovědí pacienta z dotazníku. Systém bude varovat lékaře před ztrátou těchto informací a umožní potvrdit danou operaci.

  Priorita: Střední

2.2.3 Nefunkční požadavky webového rozhraní

- **WA NFR 01 - Bezpečnost**

  Popis: Všechna rozhraní systému (včetně API pro komunikaci s mobilní aplikací pacienta) budou vyžadovat autentizaci uživatele pomocí uživatelského jména a hesla (registrace a autentizace uživatele již ve stávajícím systému existuje).

  Priorita: Vysoká

2.2.4 Popis mobilní aplikace

Webové rozhraní lékaře měla doprovázet mobilní aplikace pro pacienta, která by zobrazovala cviky v rehabilitačním plánu vytvořeného lékařem. Mělo se jednat o multiplatformní aplikaci spustitelnou jak v systému Android tak iOS.

Aplikace by na úvod umožňovala registraci zařízení v systému lékaře pomocí registračního kódu. Alternativně by aplikace umožňovala připojení k již existujícímu rehabilitačnímu plánu v systému. Po úspěšné registraci by se do úložiště aplikace uložily přihlašovací údaje a následná autentizace by probíhala automaticky.

Hlavní obrazovka aplikace by zobrazovala pokrok v rehabilitačním plánu s možností zobrazit detail jednotlivých cvičení. Aplikace by také vytvářela notifikace s upozorněním na blížící se cvičení na základě nastavení z webového rozhraní. Detail každého cvičení by obsahoval instruktážní video i textový popis cvičení s možností dokončit či přeskočit cvičení. Při dokončení cvičení by uživatel indikoval, jak se mu cvičilo vyplněním dotazníku z webového rozhraní. Uživatel by se případně mohl z aplikace odhlásit a nahrát nový rehabilitační plán.

2.2.5 Funkční požadavky mobilní aplikace

Stejně jako u webového rozhraní jsou níže strukturovaně uvedeny všechny funčkní požadavky na mobilní aplikaci pacienta.

- **MA FR 01 - Přihlášení do systému**

  Popis: Pacient si v aplikaci načte rehabilitační plán pomocí registračního kódu, který získá od lékaře. Po úspěšném spárování aplikace se systémem lékaře se do mobilního zařízení uloží přihlašovací údaje k následujícím autenztizacím.

  Priorita: Vysoká
- **MA FR 02 - Propojení již existující instance v systému**

  Popis: Aplikace umožní pacientovi se propojit s již existující instancí mobilního zařízení v systému pomocí kódu dané instance (pacient zjistí kód od lékaře).

  Priorita: Střední
- **MA FR 03 - Plán rehabilitace**

  Popis: Aplikace zobrazí pacientovi jednotlivá cvičení v rehabilitačním plánu seřazená dle data. Cviky budou označené jako:
  - Dokončené
  - Nedokončené po termínu
  - Přeskočené
  - Nedokončené před termínem
  - Aktuální pro dnešní den

  Priorita: Vysoká
- **MA FR 04 - Detail cvičení**

  Popis: Aplikace zobrazí pacientovi vybraný detail cviku z rehabilitačního plánu včetně názvu cvičení, instruktážního videa, textového popisu cvičení a statusu dokončení.

  Priorita: Vysoká
- **MA FR 05 - Dokončení cvičení**

  Popis: Aplikace umožní pacientovi označit cvičení jako (viz WA FR 12):
  - Dokončené bez obtíží
  - Dokončené s obtížemi
  - Přeskočeno

Priorita: Vysoká

- **MA FR 06 - Vyplnění dotazníku**

  Popis: V případě, že pacient dokončí cvičení s obtížemi, aplikace umožní pacientovi vybrat druh obtíže z nabídky ze systému lékaře (viz WA FR 13) nebo pacient poskytne vlastní odpověď. Odpověď se odešle do systému lékaře.

  Priorita: Vysoká

- **MA FR 07 - Notifikace**

  Popis: Aplikace upozorní pacienta na nadcházející cvičení pomocí notifikace odeslané ze systému lékaře. Čas notifikace zvolí lékař (viz WA FR 09).

  Priorita: Vysoká

- **MA FR 08 - Odhlášení**

  Popis: Aplikace umožní pacientovi se odpojit od instance v systému lékaře a načíst jinou rehabilitaci (viz MA FR 01, MA FR 02)

  Priorita: Vysoká

2.2.6 Nefunkční požadavky mobilní aplikace

- **MA NFR 01 - Bezpečnost**

  Popis: Aplikace bude komunikovat se systémem lékaře přes HTTPS s *Basic Authentication* [11].

  Priorita: Vysoká

- **MA NFR 02 - Design**

  Popis: Uživatelské rozhraní aplikace bude minimalistické a přizpůsobené k používaní seniory (velká tlačítka, velký font, max 4 barvy).

  Priorita: Střední

- **MA NFR 03 - Použitelnost**

  Popis: Navigace v aplikaci bude mít maximální hloubku zanoření 2.

  Priorita: Střední

2.3 Případy užití

Jak již ze zjištěných požadavků vyplývá, se systémem budou interagovat dva typy aktérů. Prvním aktérem je lékař používající webové rozhraní pro správu cviků a rehabilitačních plánů. Druhým aktérem je pacient používající mobilní aplikaci, přes kterou interaguje s vytvořeným rehabilitačním plánem. Zatímco lékař používá webové rozhraní nezávisle na pacientovi, pacientova aplikace po registraci v systému slouží spíše jako pouhý konzument informací od lékaře. Níže jsou popsány případy užití pro oba aktéry.

2.3.1 Případy užití pro lékaře

- **UC 01 - Vytvořit kategorii cviků**

  1. Lékař přihlášený do systému v podstránce cviků zvolí vytvoření nové kategorie
  2. Systém zobrazí formulář pro vyplnění názvu kategorie
  3. Lékař vyplní název a odešle formulář
  4. Systém vytvoří novou kategorii a zobrazí ji v seznamu v podstránce cviků

- **UC 02 - Přejmenovat kategorii cviků**
  1. Lékař přihlášený do systému v podstránce cviků zvolí kategorii, kterou chce přejmenovat
  2. Systém zobrazí stránku s detailem dané kategorie
  3. Lékař zvolí přejmenování kategorie
  4. Systém zobrazí formulář pro vyplnění nového názvu kategorie
  5. Lékař vyplní nový název a odešle formulář
  6. Systém uloží nový název kategorie a aktualizuje načtenou stránku
- **UC 03 - Smazat kategorii cviků**
  1. Lékař přihlášený do systému v podstránce cviků zvolí kategorii, kterou chce smazat
  2. Systém zobrazí hlášku s upozorněním a možností zrušení akce
  3. Lékař potvrdí smazání
  4. Systém smaže vybranou kategorii a její cviky s nově nahranými videi přesune do kategorie představující koš. Cviky, které byly vytvořeny přepoužitím již nahraných videí, smaže trvale.
- **UC 04 - Nahrát nové cvičení**
  1. Lékař přihlášený do systému v podstránce cviků zvolí kategorii, do které chce nahrát nové cvičení.
  2. Systém zobrazí detail kategorie.
  3. Lékař zvolí nahrání nového cvičení.
  4. Systém lékaři nabídne vytvoření cvičení z videí v databázi nebo nahrání nového videa.
  5. Pokud lékař zvolí nahrání videa z databáze, systém zobrazí všechna videa v databázi a lékař zvolí video, které chce přepoužít.
  6. Systém zobrazí detail cvičení s formulářem pro název a popis cvičení (a případně již vybrané video z databáze).
  7. Lékař vyplní formulář, případně nahraje nový videosoubor a uloží změny.
  8. Systém vytvoří nové cvičení ve vybrané kategorii s informacemi z formuláře a aktualizuje detail cvičení včetně náhledu nahraného videa.
- **UC 05 - Upravit cvičení**
  1. Lékař přihlášený do systému v detailu kategorie zvolí cvičení, které chce upravit.
  2. Systém zobrazí formulář pro úpravu názvu, popisu a videa cvičení.
  3. Lékař upraví jednotlivé položky a odešle formulář.
  4. Systém aktualizuje cvičení na základě informací z formuláře a aktualizuje detail cvičení.
- **UC 06 - Smazat cvičení**
  1. Lékař přihlášený do systému v detailu kategorie zvolí cvičení, které chce smazat.
  2. Systém zobrazí hlášku s upozorněním a umožní zrušit akci.
  3. Lékař potvrdí smazání.
  4. Systém přesune vybrané cvičení do kategorie představující koš. Pokud cvičení bylo vytvořeno přepoužitím jiného videa, systém cvičení smaže trvale.
- **UC 07 - Přesunout cvičení**
  1. Lékař přihlášený do systému v detailu kategorie zvolí cvičení, které chce přesunout do jiné kategorie.
  2. Systém zobrazí výběr kategorií ze seznamu.
  3. Lékař zvolí kategorii, do které chce cvičení přesunout.
  4. Systém přesune vybrané cvičení do zvolené kategorie.

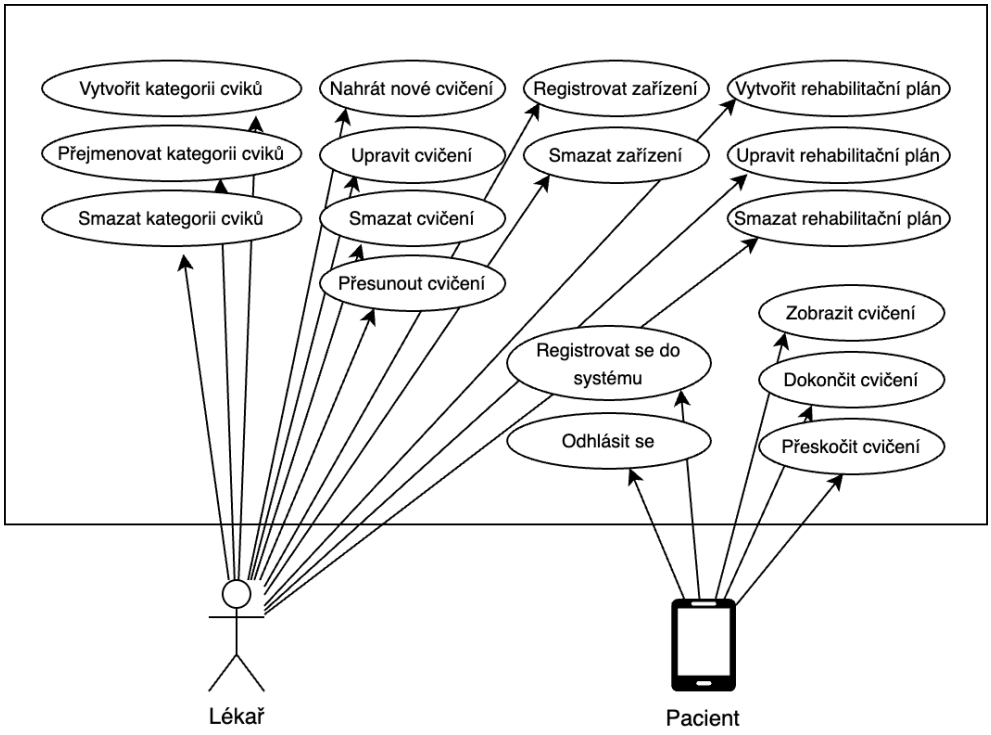
- **UC 08 - Registrovat nové zařízení pacienta**
  1. Lékař přihlášený do systému v podstránce zařízení zvolí registraci nového zařízení.
  2. Systém zobrazí náhodně vygenerovaný 4místný kód.
  3. Lékař zadá vygenerovaný kód do úvodní obrazovky aplikace pacienta a potvrdí registraci.
  4. Aplikace pacienta odešle kód do systému, který ověří jeho správnost. Pokud kód není správný, aplikace zobrazí chybovou hlášku. V opačném případě systém registruje nové zařízení a přidělí mu identifikační kód, který se uloží i do aplikace pacienta pro budoucí autentizaci. Systém lékaři zobrazí nově přidané zařízení v seznamu všech zařízení.
- **UC 09 - Smazat zařízení pacienta**
  1. Lékař přihlášený do systému v podstránce zařízení zvolí zařízení, které chce smazat.
  2. Systém zobrazí hlášku s upozorněním ztráty informací o rehabilitačních plánech na daném zařízení a nabídne zrušení akce.
  3. Lékař potvrdí smazání.
  4. Systém smaže zařízení včetně připojených rehabilitačních plánů.
- **UC 10 - Vytvořit nový rehabilitační plán**
  1. Lékař přihlášený do systému v podstránce zařízení zvolí zařízení, ve kterém chce vytvořit nový rehabilitační plán.
  2. Systém zobrazí detail zařízení.
  3. Lékař zvolí vytvoření nového rehabilitačního plánu.
  4. Systém zobrazí nový prázdný plán s formulářem pro přidávání cviků do plánu.
  5. Lékař vyplní datum nového cvičení a čas notifikace a poté vybere cvičení ze seznamu kategorií cvičení.
  6. Systém zařadí nové cvičení do rehabilitačního plánu.
- **UC 11 - Upravit rehabilitační plán**
  1. Lékař přihlášený do systému v detailu zařízení zvolí rehabilitační plán, který chce upravit.
  2. Systém zobrazí rehabilitační plán.
  3. Lékař zvolí cvičení v rehabilitačním plánu.
  4. Systém zobrazí detail cvičení s formulářem pro změnu datumu, času notifikace a samotného cviku.
  5. Lékař upraví položky ve formuláři, případně zvolí odebrání cvičení z plánu.
  6. Systém aktualizuje rehabilitační plán.
- **UC 12 - Smazat rehabilitační plán**
  1. Lékař přihlášený do systému v detailu zařízení zvolí rehabilitační plán, který chce smazat.
  2. Systém zobrazí hlášku s upozorněním a umožní akci zrušit.
  3. Lékař potvrdí smazání plánu.
  4. Systém trvale smaže rehabilitační plán ze zařízení.

2.3.2 Případy užití pro pacienta

- **UC 13 - Registrovat se do systému**
  1. Pacient na úvodní obrazovce aplikace zadá 4místný registrační kód získaný od lékaře. Pokud se chce pacient připojit k již existující instanci zařízení v systému, zadá i identifikační kód této instance.
  2. Aplikace odešle registrační kód k ověření v systému. Pokud je kód správný, aplikace zobrazí obrazovku s aktuálním rehabilitačním plánem.

- **UC 14 - Zobrazit cvičení**
  1. Pacient vybere cvičení z rehabilitačního plánu, případně zvolí možnost pro zobrazení dnešního cvičení.
  2. Systém zobrazí novou obrazovku s detailem cvičení. Obrazovka bude obsahovat video přehrávač pro instruktážní video, název cvičení a textový popis.
  3. Pacient spustí video a začne cvičit.
- **UC 15 - Dokončit cvičení**
  1. Pacient zvolí dokončení cvičení v detailu daného vičení.
  2. Systém zobrazí obrazovku s dotazníkem ohledně obtíží při cvičení.
  3. Pacient vybere, zda se mu cvičilo s obtížemi nebo bez.
  4. Pokud pacient uvede, že měl určité obtíže při cvičení, systém zobrazí pacientovi výběr ze seznamu obtíží. Pokud pacient zvolí jinou obtíž, systém zobrazí textové pole pro popsání problému vlastními slovy.
  5. Pacient potvrdí dokončení cvičení.
  6. Aplikace odešle informace o dokončení cvičení do systému a zobrazí obrazovku rehabilitačního plánu.
- **UC 16 - Přeskočit cvičení**
  1. Pokud je určité cvičení nedokončené a po termínu, aplikace v detailu cvičení poskytne možnost ho přeskočit.
  2. Pacient zvolí přeskočení cviku.
  3. Aplikace odešle informaci o přeskočení cvičení do systému a zobrazí obrazovku rehabilitačního plánu.
- **UC 17 - Odhlásit se**
  1. Pacient zvolí odhlášení na obrazovce nastavení aplikace.
  2. Aplikace smaže z lokálního úložiště autentizační kód zařízení a zobrazí úvodní obrazovku s registračním formulářem.

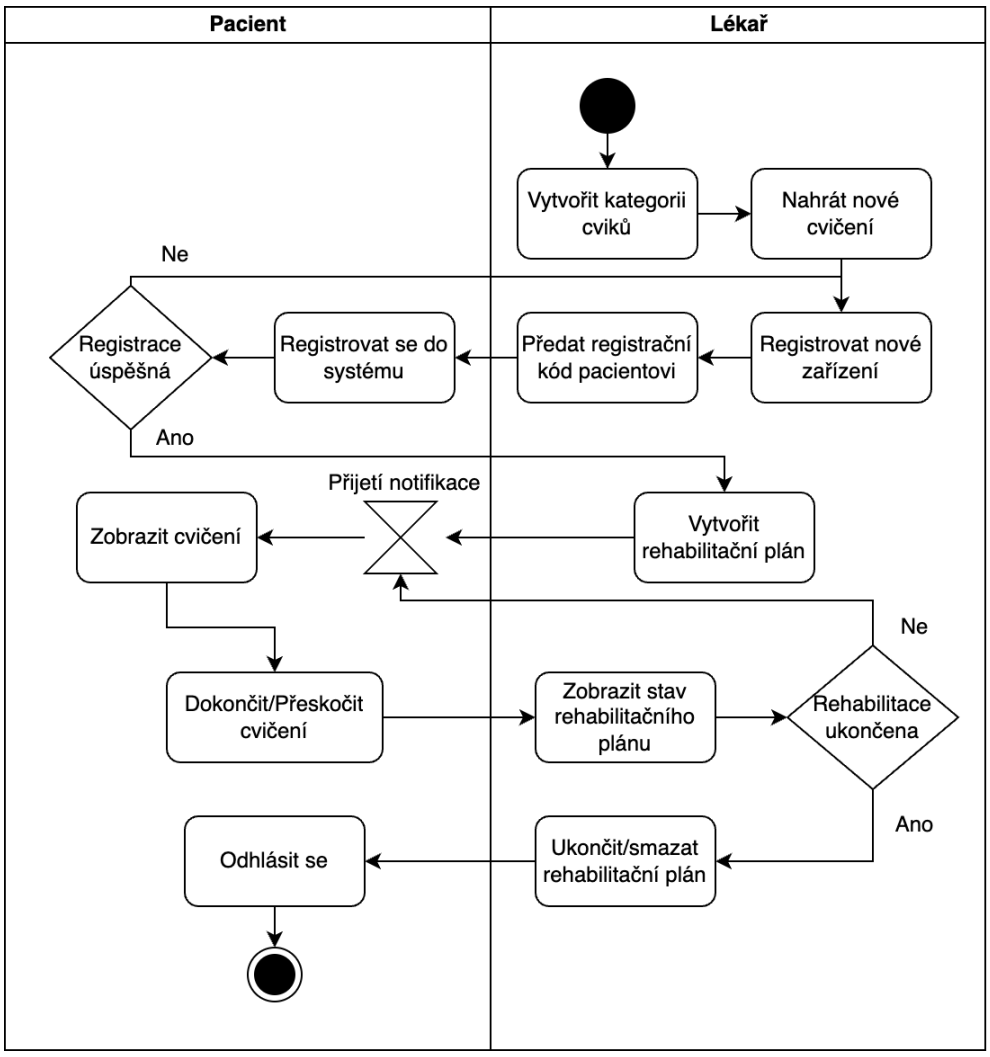
Zmíněné příklady užití jsou zobrazeny na UML Use Case diagramu 2.1.



Obrázek 2.1. Use Case diagram znázorňující případy užití systému lékařem a pacientem.

2.3.3 Proces rehabilitace

Na diagramu je zachycen celkový minimální proces rehabilitace v systému (tj. bez úprav cviků a rehabilitačních plánů). Proces se sestává z aktivit lékaře a pacienta, které jsou popsány v případech užití v předchozích sekcích.



Obrázek 2.2. Diagram aktivit znázorňující proces rehabilitace.

Kapitola 3
Architektura a design

Tato kapitola popisuje návrh struktury, implementace a nasazení vyvíjeného systému. Nejprve je navržen doménový model systému, následně je popsána jeho architektura včetně jednotlivých komponent a na závěr jsou uvedeny navržené frameworky pro implementaci architektury a diagram nasazení.

3.1 Doménový model

Na základě zjištěných funkčních požadavků na webové rozhraní i mobilní aplikaci bylo navrženo 7 základních entit, se kterými bude systém pracovat. Níže jsou jednotlivé entity popsány včetně jejich vlastností a vzájemných vztahů.

- **ExerciseCategory**
  Entita představuje kategorii cvičení zmíněnou ve funkčním požadavku WA FR 01 (viz 2.2.2). Mezi její atributy patří název kategorie a označení kategorie představující koš smazaných cviků (atribut `isBin`). Tuto entitu vytváří a spravuje jedna skupina uživatelských účtů (lékařů) v aplikaci. Každá kategorie obsahuje několik nebo žádné cvičení.
- **Exercise**
  Entita představuje cvičení popsané ve funkčním požadavku WA FR 02 v sekci 2.2.2. Entita vlastní 3 atributy:
  - `name` - Název cvičení.
  - `instructions` - Textový popis cviku doprovázející instruktážní video.
  - `videoFile` - Odkaz na soubor obsahující instruktážní video.

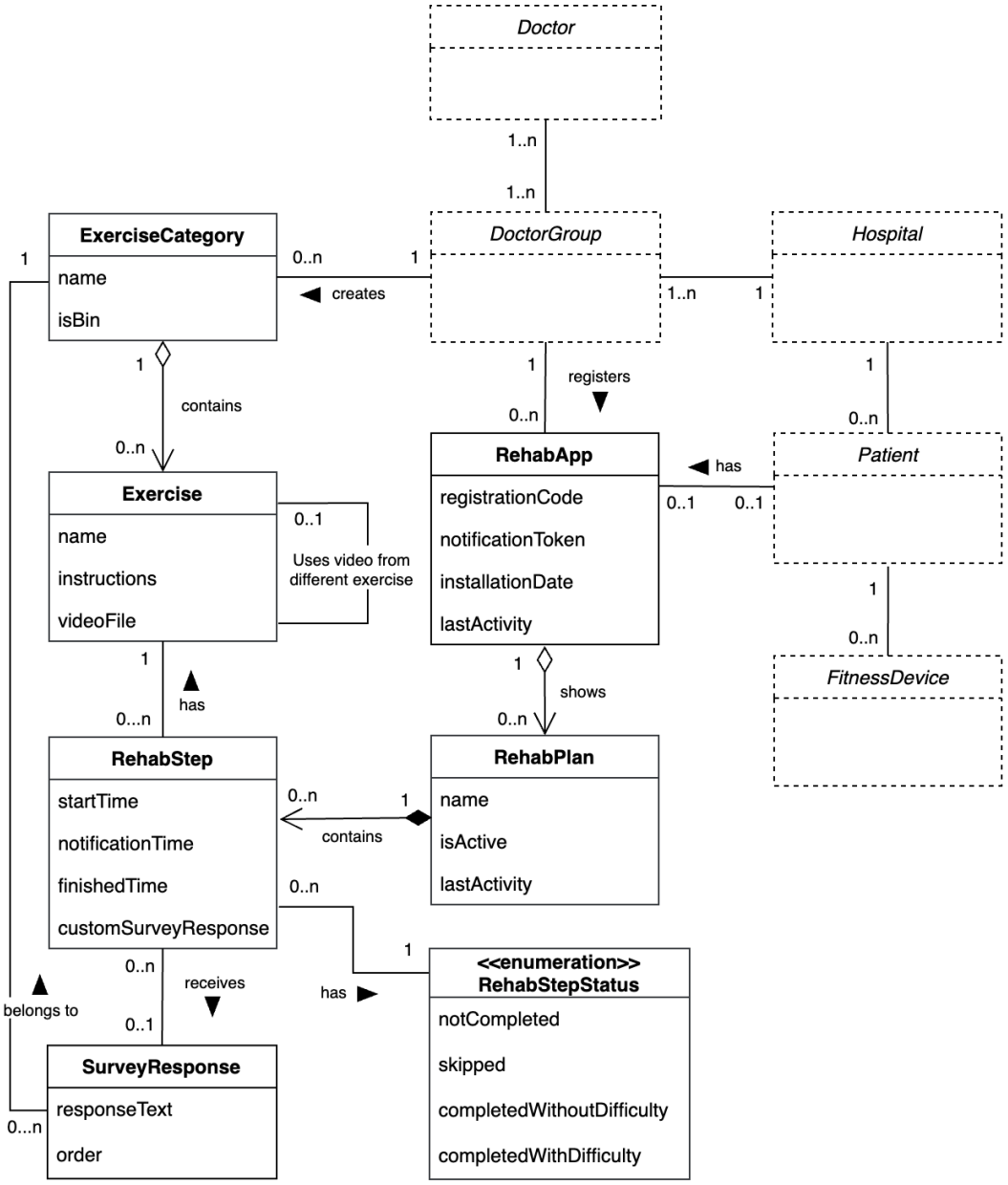
  Každé cvičení patří právě do jedné kategorie. Cvičení může a nemusí být použito v rámci několika rehabilitačních plánů. Podle požadavku WA FR 04 lze vytvořit nové cvičení přepoužitím videa z jiného cvičení, proto může tato entita obsahovat reflexivní vazbu (viz diagram 3.1).
- **RehabApp**
  Entita představuje zařízení (mobilní aplikaci) pacienta. Zařízení je registrováno do systému právě jedním lékařem (viz WA FR 06 v sekci 2.2.2) a může obsahovat několik nebo žádný rehabilitační plán. Mezi atributy entity patří:
  - `registrationCode` - Identifikační kód zařízení vygenerovaný systémem při registraci.
  - `notificationToken` - Token, který mobilní aplikace při registraci ukládá do systému pro následné odesílání notifikací ze systému do zařízení.
  - `installationDate` - Datum, kdy bylo zařízení registrováno do systému.
  - `lastActivity` - Datum a čas poslední aktivity pacienta. Poslední aktivitou se myslí čas ukončení posledního cviku v mobilní aplikaci pacienta.
- **RehabPlan**
  Entita představuje rehabilitační plán pacienta (viz WA FR 07 v 2.2.2). K danému zařízení pacienta se může vztahovat více rehabilitačních plánů, ovšem pouze jeden z nich může být aktivní (tzn. zobrazí se pacientovi v mobilní aplikaci). Každý

rehabilitační plán se skládá z několika kroků (entita `RehabStep`. Mezi atributy entity patří:

- `name` - Název rehabilitace.
- `isActive` - Označení aktivního (tj. právě probíhajícího) plánu.
- `lastActivity` - Datum a čas poslední aktivity pacienta. Poslední aktivitou se myslí čas ukončení posledního cviku v rámci rehabilitačního plánu pacienta.

RehabStep

Entita představuje krok v rehabilitačním plánu pacienta (viz WA FR 08 v 2.2.2). Protože jednotlivé kroky tvoří samotný plán, jejich existence mimo rehabilitační plán postrádá smysl. Proto je tato entita spojena s entitou `RehabPlan` kompozicí (viz diagram 3.1).



Obrázek 3.1. Doménový model systému.

Entita `RehabStep` má tyto atributy:

- `startTime` - Datum a čas naplánovaného cvičení.
- `notificationTime` - Datum a čas upozornění na nadcházející cvičení.
- `finishedTime` - Datum a čas dokončení cvičení pacientem.
- `customSurveyResponse` - V tomto atributu je uložena pacientova odpověď z dotazníku, pokud tuto odpověď napíše vlastními slovy (tzn. nevybere odpověď z nabídky).
- `RehabStepStatus` - Jedná se o výčtový typ představující stav dokončení cvičení (cvičení dokončeno, přeskočeno, dokončeno bez obtíží/s obtížemi).

Každý krok v plánu má přidružené právě jedno cvičení (entita `Exercise`) a jednu nebo žádnou (v případě, že pacient cvičil bez obtíží) odpověď z dotazníku.

- **SurveyResponse**

  Entita představuje nabízenou odpověď v dotazníku na konci cvičení (viz požadavek WA FR 12 v sekci 2.2.2). Kromě samotného obsahu má každá odpověď i své pořadí v nabídce (atribut `order`).

3.2 Architektura

Současná webová aplikace, do které je implementován tento nový modul pro rehabilitační cvičení, má *vícevrstvou architekturu* [12]. Vícevrstvé architektury patří k nejčastěji používaným architektonickým stylům v softwarovém inženýrství. Tato architektura je založená na rozdělení systému do několika oddělených horizontálních vrstev, které dohromady tvoří funkční softwarový celek.

Každá vrstva združuje komponenty (části kódu), které mají podobnou funkci v systému (tzv. *separation of concerns*). Tímto se celý systém dekomponuje na izolované části, které lze snadno nezávisle na sobě udržovat, testovat a škálovat. Většinou platí, že každá vrstva komunikuje pouze s vrstvou přímo pod sebou, čímž se snižuje provázanost mezi komponentami (pricip *high cohesion, low coupling*).

Ačkoli tato architektura nemá předepsaný přesný počet vrstev, vždy obsahuje vrstvu, která vytváří uživatelské rozhraní, vrstvu, která má na starost výpočetní část aplikace a vrstvu, která zpracovává data. Níže jsou popsány typické vrstvy této architektury:

- **Prezentační vrstva**

  Prezentační vrstva je nejvyšší vrstva aplikace, která uživateli zprostředkovává interakci se systémem. Prezentační vrstva vytváří uživatelské rozhraní, ve kterém graficky prezentuje data z business vrstvy a poskytuje formulářové prvky pro vstup uživatele. Data poskytnutá uživatelem jsou poté prezentační vrstvou zvalidována a přeposlána zpět do business vrstvy ke zpracování.

  Součástí prezentační vrstvy vyvíjené aplikace pro rehabilitační cvičení budou jak podstránky ve webovém rozhraní lékaře pro správu cviků a rehabilitačních plánů, tak obrazovky v mobilní aplikaci pacienta.

- **Business vrstva**

  Business vrstva implmentuje business logiku aplikace. V této vrstvě jsou definována pravidla a procesy pro zpracování dat z prezentační vrstvy. Tato vrstva také zajišťuje bezpečnost, logování a zpracování vzniklých výjimek při běhu aplikace. Pro práci s entitami systému v rámci business procesů tato vrstva komunikuje s nižší doménovou vrstvou.

Většina business vrstvy bude implementována ve webové aplikaci lékaře, kde se budou zpracovávat požadavky na správu cviků a rehabilitací. V této vrstvě bude také implmenetována komunikace mezi webovou a mobilní aplikací.

- **Doménová vrstva**

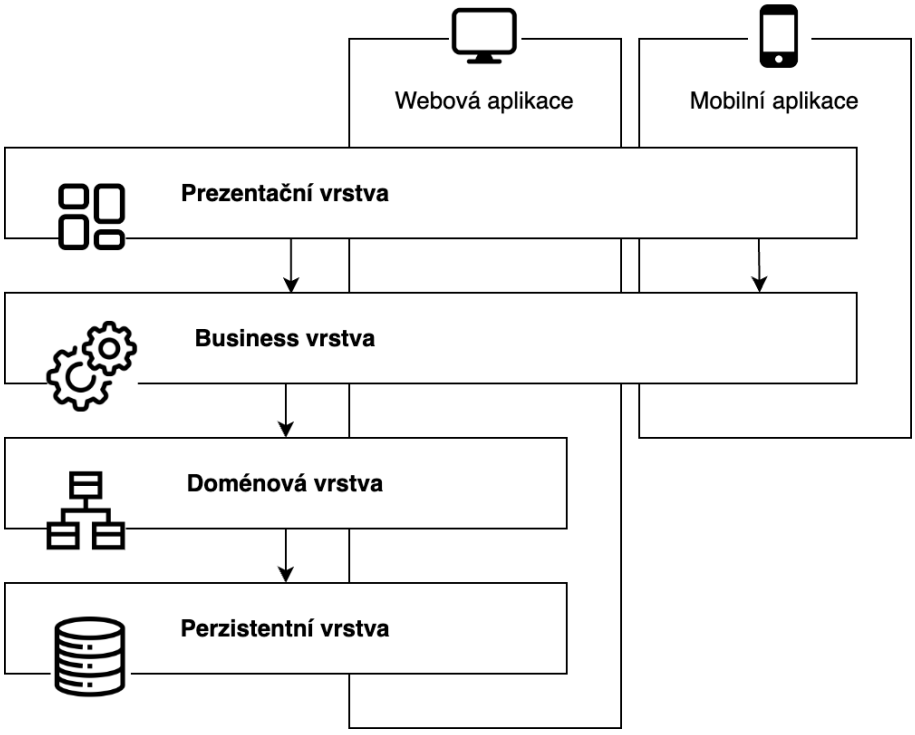
  Doménová vrstva (někdy nazývaná modelová vrstva) definuje všechny používané business entity v systému. Pro každou entitu jsou definovány její vlastnosti, metody, které poskytují a vztahy s dalšími entitami. Doménová vrstva také implementuje pravidla pro chování těchto entit v různých fázích jejich životních cyklů (vznik, update, smazání).

  Doménová vrstva bude implementována pouze ve webové aplikaci na základě doménového modelu, který je popsán v sekci 3.1.

- **Perzistentní vrstva**

  Perzistentní (nebo databázová) vrstva aplikace zajišťuje správu dat, které je potřeba trvale ukládat v databázi. Perzistentní vrstva je obvykle implementována *databázovým management systémem* (DBMS), který poskytuje ukládání a načítání dat z databáze, správu transakcí a zajištění konzistence dat.

Na obrázku 3.2 je znázorněna struktura vícevrstvé architektury ve vyvíjeném systému.



Obrázek 3.2. Struktura vícevrstvé architektury.

Mezi výhody vícevrstvé architektury patří její snadná implmemntace pomocí dostupných frameworků (viz sekce 3.3). Vývojář zpravidla pouze doplňuje potřebné komponenty do příslušných vrstev a framework již zajišťuje jejich propojení a komunikaci. Díky dekompozici aplikace na dílčí vrstvy s určitým zaměřením vzniká nižší provázanost mezi komponentami, což usnadňuje jejich modifikovatelnost a testovatelnost a zvyšuje celkovou bezpečnost aplikace. Modularita této architektury také umožňuje snadnou přepoužitelnost jednotlivých komponent v různých částech aplikace, což snižuje náklady a čas na vývoj.

Mezi nevýhody této architektury patří omezení na škálovatelnost pouze ve vertikálním směru, která vyplývá ze silných závislostí mezi jednotlivými vrstvami frameworku. Další nevýhodou může být náročnější udržitelnost této architektury oproti např. architektuře *microservices*. Změna v jedné vrstvě může ovlivnit celý systém, protože všechny vrstvy dohromady tvoří jednu monolytickou aplikaci. Architektura microservices je naopak založená na rozdělení systému na samostatně vyvíjené a nasazované moduly, jejichž funkčnost na sobě navzájem nezávisí. Struktura této architektury tak umožňuje snadnou horizontální škálovatelnost i udržovatelnost jednotlivých modulů, které dohromady fungují jako celek pomocí předávání zpráv [13].

Frameworky založené na vícevrstvé architektuře jsou vhodné pro aplikace, jejichž vývoj musí být rychlý, díky jejich snadné implementaci. Kvůli horší udržovatelnosti je tato architektura vhodnější pro větší, robustnější systémy.

3.3 Frameworky a nasazení

3.3.1 Frameworky pro webové aplikace

V současné době existuje celá řada populárních frameworků pro vývoj vícevrstvých webových aplikací. Jednotlivé frameworky se liší na základě programovacích jazyků, pro které jsou určené (Java, PHP, C#, Python a další), ale také na základě škálovatelnosti, výkonu, bezpečnosti, integrovatelnosti nebo dostupné dokumentace a podpory z komunity vývojářů. Níže jsou popsány výhody a nevýhody pěti frameworků, které patří mezi ty nejpoužívanější [14].

- **Django**[1]

  Django je populární framework pro webové aplikace v jazyce Python postavený na *Model-View-Controller* [15] architektuře. Framework poskytuje vysokou abstrakci webového vývoje, která zahrnuje objektově-relační mapování pro správu databáze, autentizaci a autorizaci, URL routování, podporu pro několik šablonovacích enginů a mnoho dalšího.

  Mezi jeho výhody patří rychlost vývoje, kterou poskytuje díky již naimplementovaným funkcionalitám. Mezi ně patří bezpočnostní prvky zahrnující ochranu před častými zranitelnostmi jako jsou *SQL injection* [16] nebo *cross-site scripting* [17]. Další výhodou je vestavěné administrační rozhraní, pomocí kterého lze snadno spravovat obsah webové aplikace bez nutnosti dalšího vývoje. Django vyniká také schopností zvládat velký provoz v systému a tudíž i širokou škálou aplikací, od tvorby e-commerce webů po sociální sítě.

  Mezi nevýhody frameworku patří jeho komplexita a obtížnost učení pro začátečníky, která vyplývá z vysoké úrovně abstrakce. Implementace aplikace v Djangu vyžaduje hodně závislostí na již vestavěných funkcích, kvůli kterým je výkon frameworku nižší v porovnání s ostatními.

- **Ruby on Rails**[2]

  Ruby on Rails je dalším velmi populárním frameworkem napsaným v objektově orientovaném jazyce Ruby. Mezi společnosti, které používají tento framework, patří např. Airbnb, GitHub, nebo Shopify. Podobně jako další frameworky poskytuje správu dat pomocí objektově-relačního mapování (tzv. ActiveRecord), RESTful routování nebo schopnost generovat základní CRUD (Create, Read, Update, Delete) operace nad datovými modely.

[1] `https://www.djangoproject.com/`
[2] `https://rubyonrails.org/`

Ruby on Rails uplatňuje princip *convention over configuration*, což znamená, že urychluje vývoj vhodným výchozím nastavením aplikace bez nutnosti explicitní konfigurace. Vývoj v tomto frameworku je urychlen i díky automatickému generování kódu např pomocí tzv. scaffolding, což je funkce genrující již zmíněné CRUD operace. Framework je vysoce modulární, což umožňuje vývojářům využít pouze ty části, které potřebují, nebo doplnit vlastní funkcionality. Podobně jako u frameworku Django má i tento aktivní komunitu vývojářů, která ho nadále podporuje.

Mezi některé nevýhody patří nižší flexibilita zapříčeněná vysokým množstvím výchozích nastavení ve frameworku. Další nevýhodou jsou problémy vznikající při upgradech do nové verze frameworku, které často vyžadují značné změny v implementaci.

- **Lavarel**[3]

  Lavarel je webový framework v jazyce PHP vyvinutý v roce 2011, tedy ze zmiňovaných frameworků nejmladší. Přesto se rychle stal oblíbeným díky své elegantní a expresivní syntaxi. Podobně jako u předchozích i tento framework poskytuje objektově-relační mapování (tzv. Eloquent), šablovací engine Blade a snadné routování.

  Lavarel je velice modulární framework umožňující sestavit širokou škálu webových aplikací. Mezi přední výhody oproti dalším frameworkům patří jeho schopnost zvládnout vysoký provoz. Další výhodou je již implementovaný celý autentizační systém - přihlášení, registrace a obnovení hesla. Podobně jako u jiných frameworků i tento poskytuje ochranu před známými útoky jako je SQL injection nebo cross-site scripting. V neposlední řadě Lavarel poskytuje automatické vytváření unit testů což značně urychluje celkový vývoj.

  Protože se jedná stále o relativně nový framewrok, kód frameworku se stále vyvíjí a rychle mění. Pro vývojáře to přináší težší údržbu aplikace při updatech na novější verze frameworku. Další nevýhodou jsou místy nejasné nebo zastaralé části dokumentace.

- **ASP.NET**[4]

  ASP.NET je framework vyvíjený společností Microsoft, který umožňuje vytvářet webové aplikace v jazycích C# nebo Visual Basic.NET. Nejnovější open-source verze ASP.NET Core umožňuje vytvářet aplikace i napříč různými operačními systémy (Windows, Linux i macOS).

  Mezi hlavní výhody patří snadná integrace frameworku s dalšími nástroji Microsoftu jako je Visual Studio, Azure nebo SQL Server, což značně usnadňuje implementaci, testování i nasazení aplikace. Další výhodou je vysoká škálovatelnost a podpora vývoje aplikací s mikroservisní architekturou. Podobně jako u dalších frameworků ASP.NET poskytuje celou řadu bezpečnostních opatření a již implementovaných funkcí.

  Hlavní nevýhodou oproti dalším frameworkům je uzavřenost vývojového prostředí na produkty od Microsoftu. Některé služby spojené s vývojem a provozem ASP.NET aplikací jsou navíc zpoplatněny (např. Windows Server nebo SQL Server Permissions). Mezi další nevýhody patří horší bezpečnostní vlastnosti oproti ostatním frameworkům nebo nedostačující dokumentace.

[3] `https://laravel.com/`
[4] `https://dotnet.microsoft.com/en-us/apps/aspnet`

- **Spring**[5]

  Spring patří mezi nejpopulárnější frameworky pro vývoj enterprise aplikací v jazyce Java. Jedná se o vysoce modulární framework s podporou pro integraci dalších Java frameworků jako jsou Hibernate pro objektově-relační mapování nebo JSF (JavaServer Faces) pro tvorbu uživatelského rozhraní.

  Typickou vlastností tohoto frameworku je uplatnění principu *Inversion of Control (IoC)*, což znamená, že vývojář pouze definuje objekty nebo části kódu, které předává frameworku ke správě [18]. Tento druh vývoje odděluje vykonávání určitých úloh v programu od jejich implementace, což umožňuje vytvářet různé implementace pro stejné úlohy a usnadňuje jejich nezávislé testování. Další výhodou frameworku je jeho lehkost z hlediska velikosti aplikace, protože je implementován v POJO (Plain Old Java Object). Vývojář tak při implementaci nemusí využívat speciální třídy nebo rozhraní. Spring framework zahrnuje velké množství projektů poskytující specifické řešení pro webové aplikace (Spring Boot), datový přístup (Spring Data), zabezpečení (Spring Security), vytváření distribuovaných systémů (Spring Cloud) a mnoho dalšího.

  Kvůli tomu, že framework poskytuje více možností pro implementaci stejného mechanizmu, může docházet při vývoji ke zmatku ohledně toho, jakou možnost zvolit, přičemž špatné rozhodnutí může vývoj opozdit. Další nevýhodou může být absence konkrétních pokynů či pravidel pro implementaci bezpečných aplikací. Vývojář tak musí sám přemýšlet, jak předejít typickým zranitelnostem, jako je např. cross-site scripting.

Vzhledem k tomu, že současné webové rozhraní aplikace TERESA bylo vytvořeno ve frameworku Spring, vyvíjený modul pro rehabilitační cvičení byl implementován ve stejném frameworku v rámci stejné aplikace. Podrobný popis struktury frameworku je popsán v sekci 4.1.1 .

3.3.2 Frameworky pro mobilní aplikace

Pro vývoj mobilních aplikací v dnešní době existují dva hlavní směry: nativní a hybridní.

Nativní vývoj znamená implementaci aplikací pro určitou platfomu, jako je iOS nebo Android pomocí jejich specifických programovacích jazyků a vývojových prostředí. Pro vývoj iOS aplikací to znamená použití jazyků Swift nebo Objective-C a vývojového prostředí Xcode. Mobilní aplikace pro operační systém Adrnoid se vyvíjí v jazyce Java nebo Kotlin a standardním vývojovým prostředím je Android Studio.

Na druhé straně hybridní vývoj představuje vytváření aplikací pomocí webových technologií jako HTML, CSS a JavaScript, které jsou zabaleny v nativním kontejneru, což umožňuje, že jsou spustitelné na více platformách [19]. Hybridní aplikace jsou vyvíjeny pomocí specifických frameworků, které poskytují přístup k nativním funkcím obou platforem pomocí JavaScriptových API.

Nativní i hibridní vývoj mají své výhody i úskalí. Mezi výhody nativního přístupu patří lepší výkon a uživatelský zážitek výsledných aplikací. Díky tomu, že jsou optimalizované pro specifickou platformu a mají přímý přístup k hardwarovým i softwarovým zdrojům dané platformy, jsou zpravidla rychlejší a efektivnější (např. v práci s pamětí) než aplikace hibridní. Hibridní aplikace na druhé straně musí implementovat určitou vrstvu abstrakce pro komunikaci mezi webovým a nativním kódem, která celkový výkon zpomaluje.

[5] `https://spring.io/`

Další výhodou nativního vývoje je využití nativních komponent a návrhových vzorů uživatelského rozhraní. To uživateli přináší povědomý vzhled a zážitek a působí konzistentně se vzhledem dané platformy. Hibridní vývoj oproti tomu vyžaduje vlastní implementaci komponent uživatelského rozhraní, které budou mít typicky stejný vzhled napříč všemi platformami.

Hibridní aplikace ovšem přináší značné výhody z hlediska času a ceny vývoje a údržby. V situaci, kdy společnost potřebuje vyvíjet mobilní aplikaci pro iOS i Android, musí zaměstnávat oddělené týmy vývojařů specializovaných na obě platformy. To vyžaduje více času i prostředků na vývoj oproti multiplatformní alternativě, která je založená na vývoji v jediném programovacím jazyce nebo frameworku. Ty samé výhody platí i o následné údržbě těchto aplikací. Jedná se tedy vždy o výběr mezi kvalitou a ekonomickou složkou vývoje.

Mezi populární frameworky pro hybridní vývoj mobilních aplikací v dnešní době patří React Native, Ionic, Flutter, Cordova nebo Xamarin [20]. Srovnání prvních tří je popsáno níže.

- **React Native**[6]

  React Native je open-source framework vyvíjený společností Facebook. Tento framework je založený na JavaScriptové knihovně React[7], která se často používá pro tvorbu webových aplikací. Díky popularitě této knihovny se i React Native stává čím dál více používanějším frameworkem pro hibridní vývoj.

  React Native oproti jiným frameworkům využívá nativní komponenty uživatelského rozhraní, což se projevuje ve vyšší rychlosti a responzivitě výsledných aplikací. Další výhodou je dostupnost velkého množství knihoven a modulů třetích stran, které tento framework rozšiřují o implementace užitečných funkcionalit v rámci aplikací (např. správa notifikací, přístup k fotoaparátu, navigace v aplikaci a další). Framework také poskytuje tzv. *live reload*, což je funkce umožňující vidět okamžité změny v aplikaci během vývoje.

  Mezi nevýhody frameworku patří velikost sestavených aplikačních balíčků, která je ovlivněná Javascriptovou částí aplikace, která vykresluje výsledné uživatelské rozhraní.

- **Ionic**[8]

  Ionic je další open-source framework pro vývoj multiplatformních aplikací. Od React Native se liší primárně tím, že nevyužívá nativní UI (User Interface) komponenty, namísto toho renderuje aplikaci psanou pomocí HTML, CSS a JavaScriptu v nativní komponentě pro vykreslování webových stránek (tzv. *WebView*[9]) podobně jako webový prohlížeč.

  Výhodou frameworku jsou již předpřipravené UI komponenty, které se snaží napodobyt vzhled těch nativních. Vývojář tak nemusí ztrácet čas implementací vlastních. Díky tomu, že Ionic nevyžaduje pro implementaci žádnou specifickou JavaScriptovou knihovnu, je velice snadným nástrojem pro webové vývojáře, kteří již mají zkušenost s webovými technologiemi. Stejně jako React Native i Ionic (pomocí pluginů z dalšího WebView frameworku Cordova) poskytuje přístup k nativním funkcím mobilních zařízení.

  Mezi nevýhody frameworku patří nižší výkon výsledných aplikací. Oproti React Native jsou aplikace vyvíjené v Ionic pomalejší a méně responzivní právě kvůli

[6] https://reactnative.dev/
[7] https://react.dev/
[8] https://ionicframework.com/
[9] https://ionicframework.com/docs/core-concepts/webview

WebView, které přidává další abstrakci mezi aplikací a operačním systémem zařízení. Dalším problémem je horší integrace modulů třetích stran do frameworku.

- **Flutter**[10]

  Flutter je další populární framework vyvíjený společností Google. Oproti předchozím dvěma frameworkům využívá vlastní nový programovací jazyk Dart[11] podobající se jazykům Java nebo Swift. Hlavní charakteristikou frameworku je jeho vysoký výkon zapříčeněný vlastním renderovacím enginem optimalizovaným pro mobilní zažízení.

  Podobně jako u Ionic i Flutter poskytuje širokou škálu předpřipravených UI komponent pro snadnou tvorbu uživatelského rozhraní. Navíc stejně jako React Native i tento framewrok má funkci *hot reload* pro okmažité probublání změn do vyvíjené aplikace. Další výhodou tohoto frameworku je vývojové prostředí FlutterFlow[12], které umožňuje poskládání aplikace bez nutnosti programování. Tento nástroj poskytuje již připravené šablony uživatelského rozhraní, které stačí sestavit a přidat vlastní aplikační logiku.

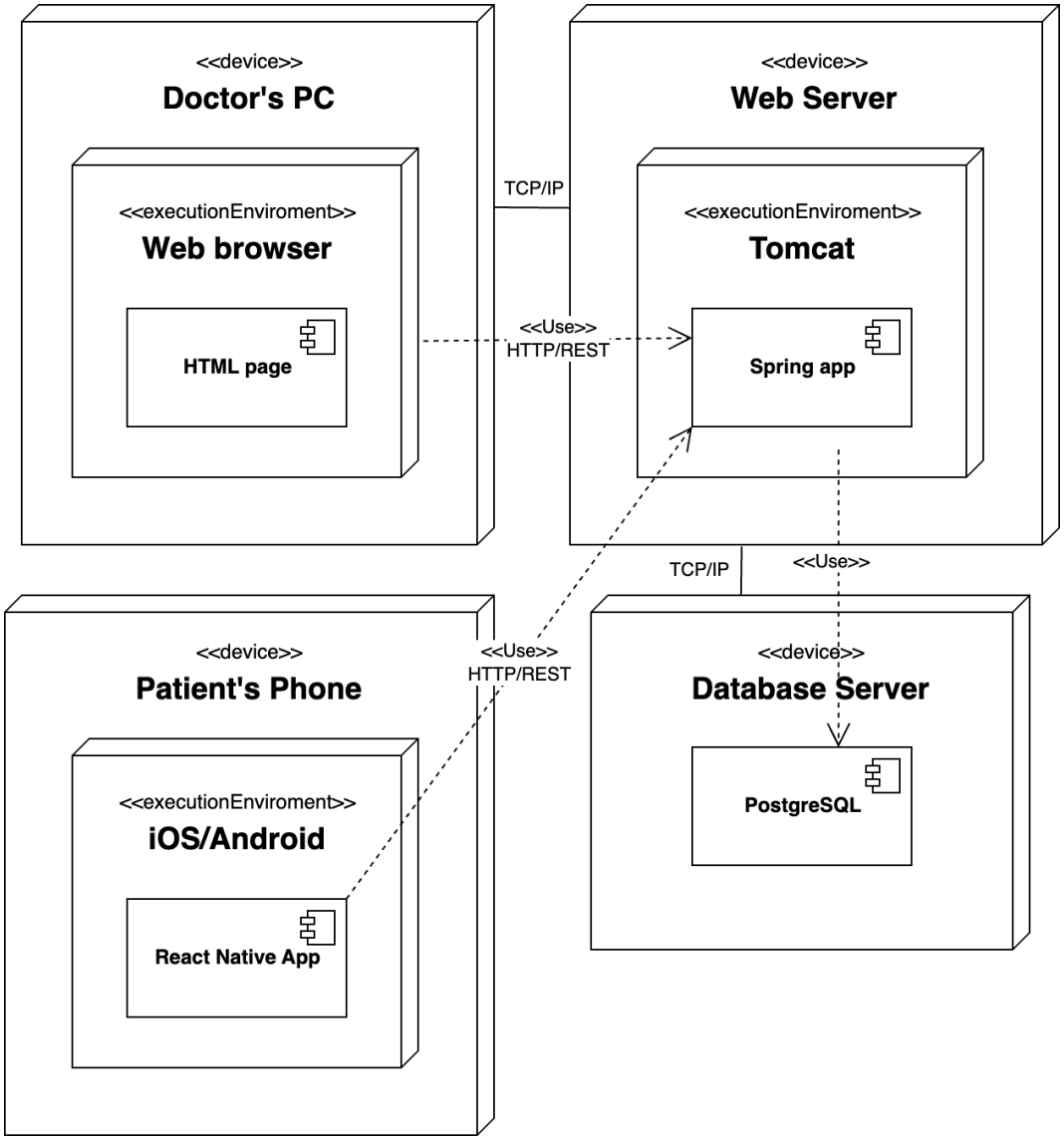
  Ačkoli jsou aplikace vyvíjené v tomto frameworku výkonné, jejich velikost značně přesahuje velikost aplikací nativních. Další nevýhodou zapříčeněnou vlastním programovacím jazykem je nekompatibilita Fultter aplikací s webovými prohlížeči. Flutter aplikace jsou tedy určené pouze pro mobilní zařízení a oproti předchozím frameworkům je nelze přepoužít pro webové aplikace. Protože se jedná stále o relativně nový framework, oproti React Native nemá tak velkou podporu knihoven třetích stran.

Pro vývoj mobilní aplikace pacienta byl zvolen hibridný způsob vývoje za použití frameworku React Native především z důvodu předchozí zkušenosti autora s tímto frameworkem. Protože se bude jednat o malou mobilní aplikaci (kolem pěti obrazovek) bez zvláštních nároků uživatele na výkon, zvolený framework bude dostačující. Navíc umožní sestavení aplikace pro iOS i Android prostředí.

3.3.3 Diagram nasazení

Na obrázku 3.3 je znázorněn diagram celkového nasazení systému na fyzická zařízení. Jak webový klient lékaře tak mobilní aplikace pacienta bude komunikovat s webovým serverem přes RESTful rozhraní [21] pomocí HTTP požadavků. Mobilní aplikace pacienta postavená ve frameworku React Native poběží v iOS či Android prostředí jako nativní spustitelná aplikace (formát .ipa pro iOS, .apk pro Android). Webová aplikace bude implementovaná ve frameworku Spring, který využívá webový server Apache Tomcat pro řízení komunikace přes HTTP protokol. Perzistentní data aplikace budou ukládána v databázi na odděleném serveru pro zvýšení bezpečnosti a škálovatelnosti systému. Současný systém již využívá relační databázi PostgreSQL pro ukládání dat z fitness náramků, proto bude právě tato databáze použita i pro nový modul v současné Spring aplikaci.

[10] `https://flutter.dev/`
[11] `https://dart.dev/`
[12] `https://flutterflow.io/`



Obrázek 3.3. Diagram nasazení webové a mobilní aplikace.

3.4 Návrh uživatelského rozhraní

3.4.1 Uživatelské rozhraní webové aplikace

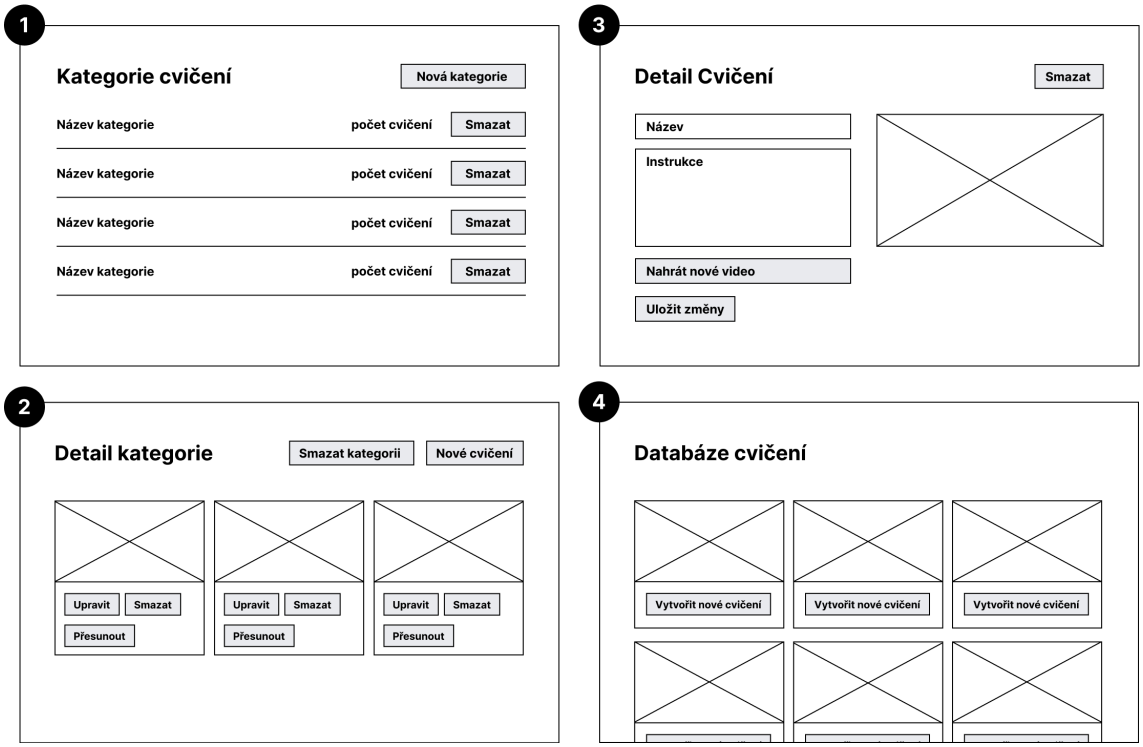
Na základě doménového modelu (viz 3.1) bylo uživatelské rozhraní pro aplikaci lékaře rozděleno na dvě hlavní podstránky: Cvičení a Rehabilitace.

Podstránka Cvičení bude umožňovat lékaři nahrávat nová cvičení a organizovat je do kategorií. Podstránka bude obsahovat tyto obrazovky:

1. **Seznam kategorií cvičení** - Zobrazí se jako úvodní obrazovka. Poskytne lékaři seznam všech kategorií včetně počtu cvičení v každé kategorii. Lékař bude moct smazat nebo vytvořit novou kategorii.
2. **Detail kategorie** - Tato obrazovka zobrazí karty všech cvičení v dané kategorii. Na kartě cvičení bude náhled instruktážního videa, název cvičení a možnost smazat, editovat nebo přesunout cvičení do jiné kategorie. Na obrazovce budou také tlačítka

pro smazání celé kategorie a přidání nového cvičení. Na obrazovce bude možnost nahrát nové cvičení nebo použít existující z databáze cvičení.

3. **Databáze cvičení** - Podobně jako u detailu kategorie zde budou karty všech nahraných cvičení včetně náhledů videí a možnosti přepoužít video do nového cvičení.
4. **Detail cvičení** - Obrazovka zobrazí formulář s vyplněným názvem cvičení a instrukcemi ke cvičení, které zde lékař může editovat. Dále zde bude zobrazeno instruktážní video s možností nahrazením videa nově nahraným.

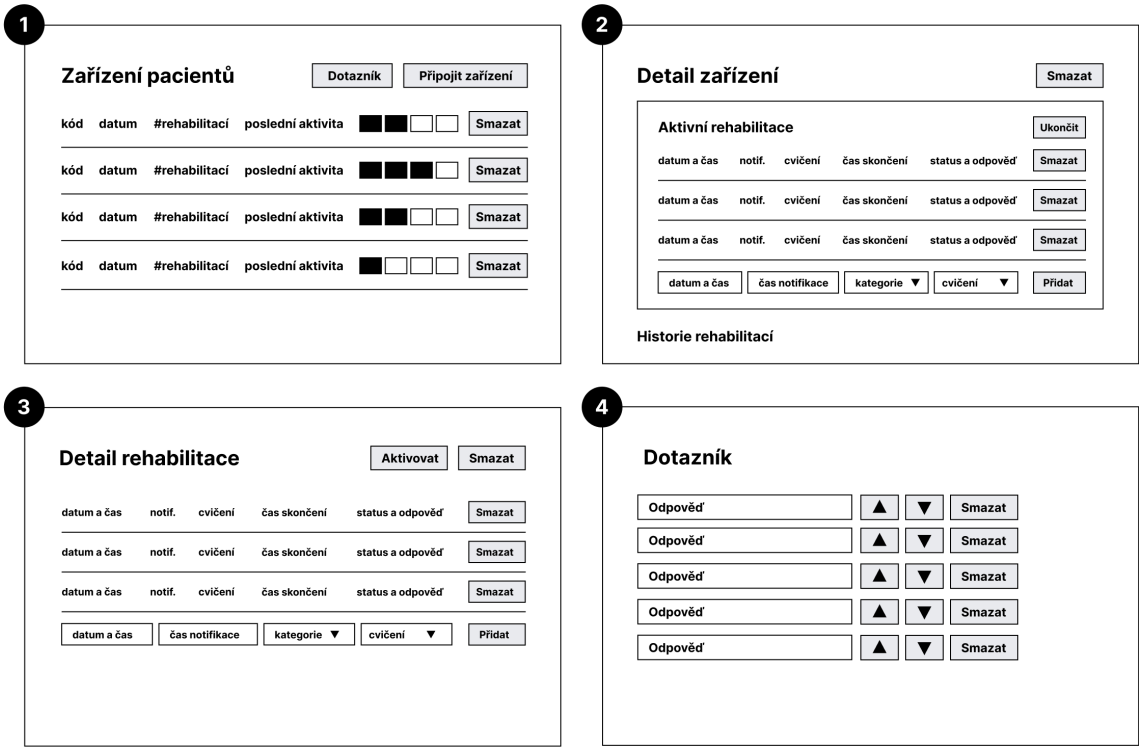


Obrázek 3.4. Wireframy obrazovek na podstránce Cvičení.

Podstránka Rehabilitace bude poskytovat přehled registrovaných zařízení pacientů a správu rehabilitačních plánů. Podstránka bude obsahovat následující obrazovky:

1. **Seznam registrovaných zařízení** - Zobrazí se jako úvodní obrazovka. Poskytne lékaři seznam všech mobilních aplikací pacientů, které byly registrovány v systémem. Lékař bude mít možnost smazat nebo připojit nové zařízení. Seznam bude obsahovat kód zařízení (mobilní aplikace), datum registrace v systému, počet proběhlých rehabilitací, poslední aktivitu pacienta a průběh aktuálního rehabilitačního plánu. Na obrazovce bude také odkaz na správu dotazníku (viz 4. obrazovka).
2. **Detail zařízení (mobilní aplikace)** - Na této obrazovce se zobrazí detail aktuálního rehabilitačního plánu (viz 3. obrazovka) a seznam již proběhlých rehabilitací. Lékař bude moci přidat nový rehabilitační plán do zařízení, smazat nebo aktivovat již proběhlé nebo ukončit právě probíhající.
3. **Detail rehabilitačního plánu** - Tato obrazovka bude zobrazovat jednotlivé kroky v rehabilitačním plánu. Každý krok bude mít datum a čas, čas notifikace, přiřazené cvičení, čas dokončení, status a odpověď z dotazníku (viz požadavek WA FR 08 v sekci 2.2.2). Lékař bude moci smazat jednotlivé kroky nebo přidat nový výběrem kategorie, poté cvičení, data, času a času notifikace.

4. **Dotazník** - Na této obrazovce bude seznam nabízených odpovědí pro dotazník pacienta po ukončení cvičení. Lékař bude mít možnost smazat nebo přidat novou odpověď nebo změnit jejich pořadí a obsah.



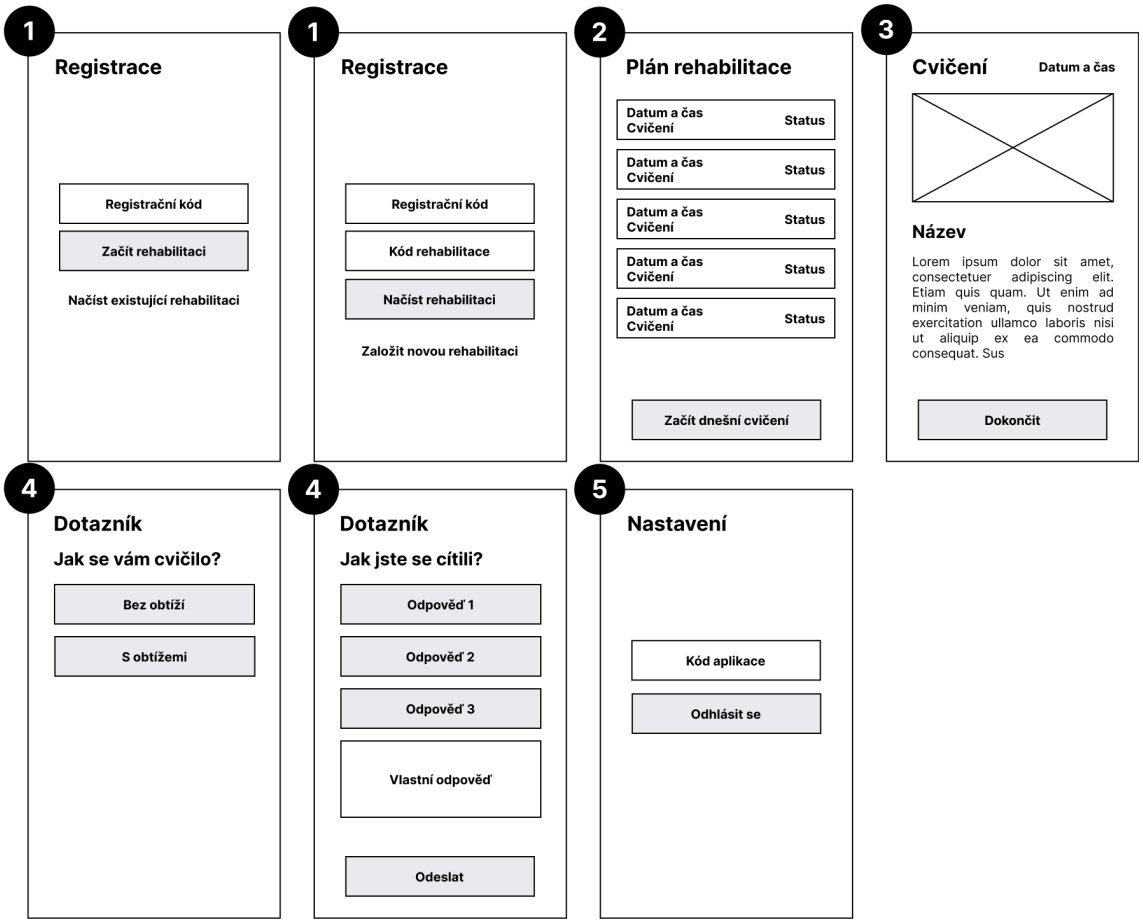
Obrázek 3.5. Wireframy obrazovek na podstránce Rehabilitace.

3.4.2 Uživatelské rozhraní mobilní aplikace

Mobilní aplikace pacienta má poskytnout registraci zařízení do systému a zobrazení aktuálního rehabilitačního plánu a jednotlivých cvičení včetně dotazníku po jejich ukončení. Uživatelské rozhraní bylo rozděleno do následujících obrazovek:

1. **Registrace** - Pokud aplikace ještě není registrovaná v systému, zobrazí pacientovi formulář pro zadání registračního kódu, který získá od svého lékaře. Na této obrazovce bude také možnost nahrát existující rehabilitační plán v systému pomocí jeho kódu (viz požadavek MA FR 02 v sekci 2.2.5).
2. **Rehabilitační plán** - Tato obrazovka se zobrazí jako úvodní, pokud bude zařízení již zaregistrované v systému. Bude zobrazovat seznam kroků v aktuálním rehabilitačním plánu. U každého kroku bude zobrazeno datum a čas cvičení, název cvičení a status (nedokončeno/přeskočeno/dokončeno). Pokud na aktuální den bude naplánováno cvičení, ve spodní části obrazovky se zobrazí tlačítko pro zobrazení tohoto cvičení (viz 3. obrazovka). Kliknutím na nějaké cvičení v seznamu se zobrazí obrazovka č. 3.
3. **Detail cvičení** - Tato obrazovka zobrazí datum a čas cvičení, název, video a popis s instrukcemi ke cvičení. Pokud pacient dané cvičení ještě nedokončil, ve spodní části obrazovky se zobrazí tlačítko pro dokončení cvičení. Kliknutím na toto tlačítko se objeví obrazovka č. 4.
4. **Dokončení cvičení** - Na této obrazovce pacient uvede, zda se při cvičení vyskytly nějaké obtíže a pokud ano, vybere jejich konkrétní podobu ze seznamu nebo napíše vlastní zprávu. Tlačítkem “Odeslat” se odpověď odešle do systému, kde se zobrazí lékaři v detailu rehabilitačního plánu.

5. **Nastavení** - Tato obrazovka bude dostupná ze všech ostatních proklikem v horním panelu aplikace. Zde bude uveden identifikační kód zařízení a bude tu možnost "odhlášení" - tedy odpojení mobilní aplikace ze systému. Po odhlášení aplikace přejde na obrazovku č. 1.



Obrázek 3.6. Wireframy obrazovek v mobilní aplikaci pacienta.

Kapitola 4
Implementace

Tato kapitola se věnuje popisu implementace jak webové tak mobilní části vyvíjeného modulu. U webové části je představen framework Spring Boot a v následujících sekcích je popsáno jeho využití v implementaci back-endové části aplikace. Dále sekce 4.1.5 popisuje implementaci front-endu aplikace pomocí šablonovacího enginu Thymeleaf.

V druhé části této kapitoly je představen framework React Native a vývojové prostředí Expo, ve kterém je vyvíjena mobilní aplikace pacienta. V následujících sekcích je vysvětlena navigace v aplikaci, komunikace s webovou částí a práce s notifikacemi.

4.1 Implementace webové aplikace

Jak již bylo zmíněno v sekci 3.2, vícevrstvá architektura frameworku Spring umožňuje vývojáři dekomponovat implementaci systému na vrstvu perzistentní, doménovou, servisní a prezentační. Následující sekce nás provází vývojem těchto vrstev počínaje od té nejnižší (perzistentní/doménové) po nejvyšší (prezentační).

4.1.1 Spring Boot

Spring Boot[1] je projekt frameworku Spring umožňující rychlý a snadný vývoj samostatně spustitelných webových aplikací. Spring Boot odstraňuje nutnost manuální konfigurace Springu pomocí složitých XML souborů tím, že automaticky prohledává závislosti v projektu a sám generuje nutné konfigurace.

Spring Boot také obsahuje vestavěné HTTP servery Tomcat, Jetty a další, které poskytují podporu pro webové technologie jako jsou Servlet, WebSocket a RESTful API. Díky vestavěnému webovému serveru lze systém vyvíjený v tomto frameworku okamžitě spustit jako samostatně běžící webovou aplikaci.

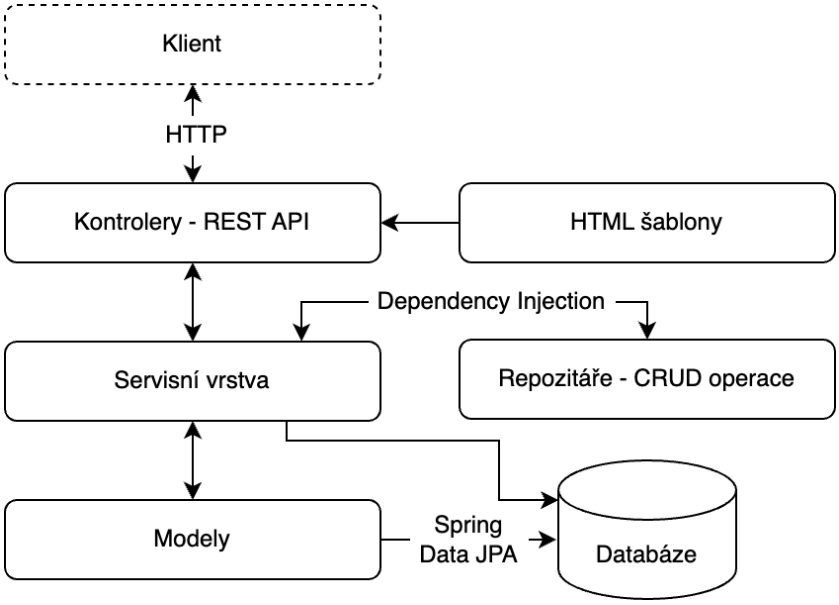
V neposlední řadě tento projekt obsahuje modul Spring Data JPA[2], který zjednodušuje implementaci perzistentní vrstvy aplikace tím, že poskytuje abstrakci nad přístupem k relační databázi. To umožňuje vývojářům snadno vytvářet tzv. repozitáře pro předpis CRUD (Create, Read, Update, Delete) operací nad daty.

Typickou architekturu Spring Boot aplikace tvoří následující vrstvy (viz obrázek 4.1):

- **Modely a repozitáře**

  Modelová vrstva se sestává z POJO (Plain Old Java Object) tříd, které reprezentují základní entity v aplikaci (viz doménový model 3.1). Tyto třídy obsahují vlastnosti daných entit a poskytují metody k jejich manipulaci. Spring Boot využívá objektově-relační mapování např. pomocí frameworku Hibernate[3], který umožňuje

[1] https://spring.io/projects/spring-boot
[2] https://docs.spring.io/spring-data/jpa/docs/current/reference/html/
[3] https://www.baeldung.com/spring-boot-hibernate



Obrázek 4.1. Architektura Spring Boot aplikace.

mapovat tyto entity definované v Java třídách na tabulky v relační databázi a naopak.

Pro přístup k databázi se využívá tzv. repozitářů. Jedná se o Java rozhraní, které definuje metody pro dotazování a manipulaci s daty v databázi. Spring Boot umožňuje automaticky generovat implementaci těchto rozhraní na základě názvů definovaných metod pomocí modulu Spring Data JPA.

- **Servisní vrstva**

  Servisní vrstva se skládá ze tříd implementujících business logiku aplikace. Tyto třídy poskytují vyšší vrstvě (kontrolerům) metody pro různé operace v aplikaci, které zahrnují vytváření, validaci, úpravu a mazázání dat. K tomu využívají rozhraní repozitářů, ke kterým přistupují pomocí principu *dependency injection*[18]. Jedná se o návrhový vzor pro komunikaci mezi dvěma objekty, který je založený na tom, že do jednoho objektu vkládáme referenci na externí třídu namísto toho, abychom ji v něm přímo instanciovali.

- **Kontrolery**

  Kontrolery jsou klíčové komponenty pro vytváření RESTful API ve Spring Boot aplikaci, jejichž úkolem je vyřizovat HTTP požadavky od klienta. Jedná se o Java třídy označené anotací `@Controller` nebo `@RestController`, které pomocí svých metod mapují jednotlivé HTTP požadavky na implementaci jejich vyřizování. Typ a URL adresa HTTP požadavků se definuje pomocí anotací jednotlivých metod (např. `@GetMapping, @PostMapping`, ...). Spring Boot navíc poskytuje další anotace jako `@RequestParam`, `@PathVariable` nebo `@RequestBody` pro přístup k datům odeslaným z formuláře, adresovým proměnným nebo k datům v těle požadavku. V odpověď na každý požadavek příslušná metoda vrací HTTP odpověď s HTML stránkou nebo XML/JSON souborem pro klientskou část aplikace.

- **Prezentační vrstva**

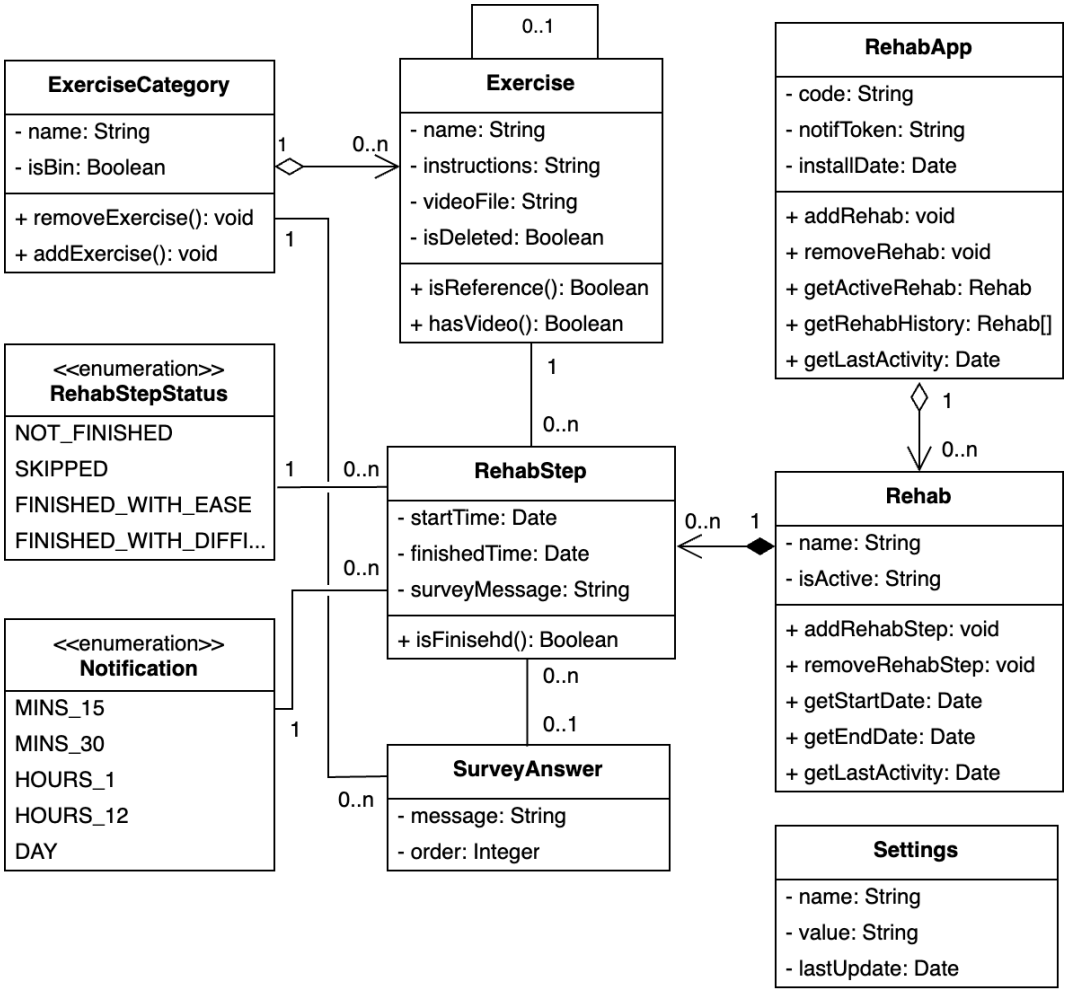
  Poslední částí architektury je prezentační vrtva, která vytváří HTML stránky pro uživatelské rozhraní klienta. Spring Boot poskytuje podporu pro řadu šablonovacích enginů jako je např. Thymeleaf, který byl zvolen pro tento projekt. Šablony Thymeleaf používají specifickou syntaxi, která umožňuje svázat data z kontrolerů

s HTML stránkami. Tato syntaxe mimo jiné zahrnuje výrazy v jazyce Java, proměnné a podmínky, díky kterým lze dynamicky generovat výsledný obsah stránek.

4.1.2 Modelová vrstva

Modelová vrstva byla implementována na základě doménového modelu pospaného v sekci 3.1. Pro každou entitu byla vytvořena Java třída obsahující příslušné atributy a vztahy. Oproti doménovému modelu byla přidána entita `Settings` pro ukládání různých nastavení v aplikaci (např. kód pro registraci pacientovy aplikace). Jednotlivé entity a jejich vztahy jsou zobrazeny v UML class diagramu 4.2.

Metody pro získávání a nastavení privátních atributů jsou v diagramu opomenuty. Díky knihovně Lombok[4] lze využít anotací `@Getter` a `@Setter`, které tyto metody pro všechny atributy automaticky generují.



Obrázek 4.2. UML class diagram modelové vrstvy aplikace.

Využitím Java Persistence API (JPA)[22] byly jednotlivé modely namapovány na tabulky v relační databázi. K tomu bylo potřeba definovat názvy příslušných tabulek (anotace `@Table`) a jejich sloupců (anotace `@Column`). Dále bylo potřeba definovat vztahy mezi entitami pomocí `@ManyToOne` a `@OneToMany` anotací. V rámci těchto

[4] `https://www.baeldung.com/intro-to-project-lombok`

anotací lze definovat i kaskádovité chování pro entity, jejichž životní cyklus závisí na jiných. Pomocí atributu `cascade` lze např. zajistit, aby v případě smazání rehabilitačního plánu došlo k smazání všech jeho kroků. Použití zmíněných JPA anotací je ilustrováno na následující ukázce kódu třídy `Rehabilitation`.

```
@Table(name = "rehabilitation", schema = "public", catalog = "p2ce")
public class Rehabilitation implements Comparable<Rehabilitation> {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "name")
    private String name;

    @Column(name = "is_active")
    private Boolean isActive = true;

    @ManyToOne
    @JoinColumn(name = "rehab_app_id", referencedColumnName = "id")
    private RehabApp rehabApp;

    @OneToMany(mappedBy = "rehabilitation",
        cascade = CascadeType.ALL,
        orphanRemoval = true)
    private List<RehabStep> steps = new ArrayList<>();
```

4.1.3 Servisní vrstva

Servisní vrstva obsahuje pro každou entitu z modelové vrstvy třídu poskytující metody pro jejich správu. K tomu tyto servisní třídy využívají JPA repozitáře, které poskytují základní CRUD operace nad databází. Jedná se o Java rozhraní, jejichž implementaci poskytuje Spring Data JPA. Velkou výhodou tohoto modulu je atomatické generování dotazů na základě názvů metod. Programátorovi tedy stačí definovat metody příslušného rozhraní (název, parametry a typ návratové hodnoty) a o implementaci je automaticky postaráno. Níže je uveden příklad repozitáře poskytující metody pro správu cvičení:

```
@Repository
public interface ExerciseRepository extends
        JpaRepository<Exercise, Long> {
    Exercise saveExercise();
    Optional<Exercise> getExerciseById(Long id);
    List<Exercise> getAll();
    void deleteExerciseById(Long id);
}
```

Do servisní vrstvy byly implementovány třídy, jejichž metody jsou popsané níže:

- **ExerciseCategoryService**
  - `getCategory` - Vrací kategorii cvičení na základě ID v parametru (pokud existuje v databázi).
  - `getAllCategories` - Vrací seznam všech kategorií v databázi.
  - `getBin` - Vrací kategorii cvičení představující koš pro smazaná cvičení s nahranými videi.
  - `saveCategory` - Uloží kategorii v parametru do databáze.
  - `deleteCategory` - Smaže kategorii v databázi na základě ID v parametru.
  - `renameCategory` - Přejmenuje kategorii s ID v prvním parametru na název uvedený v druhém parametru.
- **ExerciseService**
  - `getExercise` - Vrací cvičení na základě ID v parametru (pokud existuje v databázi).
  - `getAllExercises` - Vrací seznam všech cvičení v databázi.
  - `createExercise` - Ukládá cvičení vytvořené nahráním nového videa nebo přepoužitím videa z již existujícího cvičení v databázi.
  - `updateExercise` - Ukládá provedené změny u cvičení. Pokud došlo k nahrání nového videa, předchozí se smaže z úložiště.
  - `deleteExercise` - Smaže cvičení v databázi na základě ID v parametru. V případě, že cvičení obsahuje unikátní video, přesune se pouze do kategorie představující koš.
  - `moveExercise` - Přesune cvičení s ID v prvním parametru do kategorie s ID uvedeném v druhém parametru.
- **RehabAppService**
  - `getRehabApp` - Vrací aplikaci pacienta na základě ID v parametru (pokud existuje v databázi).
  - `getRehabAppByCode` - Vrací aplikaci pacienta na základě jejího unikátního kódu v parametru (pokud existuje v databázi).
  - `getAllRehabApps` - Vrací seznam všech aplikací pacientů v databázi.
  - `saveRehabApp` - Uloží aplikaci v parametru do databáze (vytvoří novou nebo uloží změny).
  - `deleteRehabApp` - Smaže aplikaci a její rehabilitační plány v databázi na základě ID v parametru.
- **RehabilitationService**
  - `getRehabilitation` - Vrací rehabilitační plán pacienta na základě ID v parametru (pokud existuje v databázi).
  - `getAllRehabilitations` - Vrací seznam všech plánů v databázi.
  - `saveRehabilitation` - Ukládá změny provedené v plánu v parametru.
  - `createRehabilitation` - Vytvoří nový plán v databázi předaný v parametru.
  - `deleteRehabilitation` - Smaže plán v databázi na základě ID v parametru.
  - `finishRehabilitation` - Označí plán s ID v parametru za dokončený.
  - `activateRehabilitation` - Zkontroluje, zda neexistuje již aktivní rehabilitační plán, v opačném případě označí plán s ID v parametru za aktivní.
- **RehabStepService**
  - `getRehabStep` - Vrací krok v rehabilitačním plánu na základě ID v parametru (pokud existuje v databázi).
  - `getAllRehabSteps` - Vrací seznam všech uložených kroků ze všech plánů v databázi.

- `saveRehabStep` - Uloží krok v parametru do databáze (vytvoří nový nebo uloží změny).
- `deleteRehabStep` - Smaže krok z plánu v databázi na základě ID v parametru.

- **RehabSurveyAnswerService**
  - `createAnswer` - Vytvoří novou odpověď v dotazníku z textu v parametru.
  - `getAll` - Vrací seznam všech odpovědí v dotazníku.
  - `deleteAnswer` - Smaže odpověď s ID v parametru z databáze.
  - `moveUp` - Změní pořadí odpovědi s ID v parametru o jeden stupeň výše.
  - `moveDown` - Změní pořadí odpovědi s ID v parametru o jeden stupeň níže.
  - `renameAnswer` - Přepíše odpověď s ID v prvním parametru na text uvedený v druhém parametru.
- **SettingService**
  - `getSetting` - Vrací systémové nastavení dle ID v parametru (pokud existuje v databázi).
  - `getRegCodeSetting` - Vrací 4místný kód pro registraci nové aplikace pacienta v systému. Pokud žádný kód není v databázi, vygeneruje nový.
  - `updateRegCodeSetting` - Náhodně vygeneruje a uloží do databáze nový 4místný registrační kód.
  - `saveSetting` - Uloží nastavení předané v parametru.
  - `deleteSetting` - Smaže nastavení v parametru z databáze.
- **StorageService**
  - `init` - Vytvoří v kořenovém adresáři nový adresář (pokud již neexistuje), do které se budou ukládat nahrávaná videa.
  - `store` - Uloží do adresáře nahrané video (objekt `MultipartFile` ve Springu) a přejmenuje na název v parametru.
  - `load` - Vrátí cestu k uloženému videu na základě jeho názvu v parametru.
  - `delete` - Smaže video z adresáře na základě jeho názvu v parametru.

4.1.4 RESTful API

RESTful API je aplikační programovací rozhraní, které je postavené na principech architektonického stylu REST (Representational State Transfer) pro webové služby [21]. Tento styl reprezentuje zdroje (data nebo funkce aplikace) pomocí URL adres a přistupuje k nim nebo jimi manipuluje pomocí HTTP metod (GET, POST, PUT, DELETE). RESTful API implementují bezstavovou klient-server architekturu, což znamená, že si server mezi jednotlivými požadavky klienta neudržuje jeho stav.

Jak bylo zmíněno v sekci 4.1.1, Spring Boot poskytuje anotace (`@Controller`, `@GetMapping`, ...) pro označení tříd, jejichž metody vyřizují jednotlivé HTTP požadavky. Ve vyvíjeném systému metody těchto tříd (tzv. kontrolery) vrací buď HTML stránky vytvořené pomocí šablonovacího enginu Thymeleaf nebo JSON objekty pro komunikaci s mobilní aplikací klienta. Níže je ukázka implementace metody `getRehab`, která vrací pro konkrétní mobilní aplikaci pacienta jeho aktivní rehabilitační plán.

```
@GetMapping(value = "/rehab-app/{id}/active-rehab",
            produces="application/json")
public ResponseEntity<RehabilitationDTO>
        getRehab(@PathVariable Long id) {
    RehabilitationDTO response;
    HttpStatus status;
    Optional<RehabApp> rehabApp = rehabAppService.getRehabApp(id);
```

```
    if(rehabApp.isPresent()) {
        response = rehabApp.get().toDTO().getActiveRehab();
        status = HttpStatus.OK;
    } else {
        response = new RehabilitationDTO();
        status = HttpStatus.NOT_FOUND;
    }
    return new ResponseEntity<>(response, status);
}
```

Anotace `@GetMapping` říká, že tato metoda bude zpracovávat GET požadavky na adresu `/rehab-app/{id}/active-rehab`. Pomocí anotace `@PathVariable` lze získat adresovou proměnnou `id`, která představuje identifikátor mobilní aplikace. Následně se volá metoda servisní třídy `rehabAppService`, která vrátí entitu mobilní aplikace dle identifikátoru z databáze. Pomocí objektu `ResponseEntity` pak tato metoda vrací aktivní rehabilitační plán příslušné mobilní aplikace v HTTP odpovědi se stavovým kódem 200 (OK) nebo 404 (Not Found).

V následující tabulce 4.3 je uveden přehled všech implementovaných kontrolerů a jejich RESTful API, které vystavují.

Kontroler	URL	Metoda	Popis
ExerciseCategory Controller	/exercise-category	GET	Zobrazí výpis kategorií cvičení (šablona **exercise-categories.html**)
		POST	Uloží novou kategorii
	/exercise-category/{id}	GET	Zobrazí detail kategorie (šablona **exercise-category.html**)
		PUT	Přejmenuje kategorii
		DELETE	Smaže kategorii
ExerciseController	/exercise/{id}	GET	Zobrazí detail cvičení (šablona **exercise.html**)
	/exercise/	POST	Uloží nové cvičení nebo změny ve cvičení
	/exercise/{id}	DELETE	Smaže cvičení
	/exercise/{id}/move	PUT	Přesune cvičení do jiné kategorie
	/exercise/new	GET	Zobrazí detail nového cvičení (šablona **exercise.html**)
RehabAppController	/rehab-app	GET	Zobrazí seznam všech zařízení pacientů (šablona **rehab-apps.html**)
	/rehab-app/{id}	GET	Zobrazí detail zařízení (šablona **rehab-app.html**)
		DELETE	Smaže (odpojí) zařízení
RehabController	/rehab/{id}	GET	Zobrazí detail rehabilitačního plánu (šablona **rehab-app.html**)
	/rehab-app/{id}/new-rehab	POST	Vytvoří nový rehabilitační plán
	/rehab/{rehab_id}/new-step	POST	Přidá nový krok do rehab. plánu

	/rehab-step/{step_id}	PUT	Upraví krok v rehab. plánu
		DELETE	Smaže krok v rehab. plánu
	/rehab/{rehab_id}/finish	PUT	Ukončí rehab. plán (deaktivuje)
	/rehab/{rehab_id}/activate	PUT	Aktivuje rehabilitační plán z historie
RehabAnswer Controller	/rehab-answer	GET	Zobrazí seznam odpovědí v dotazníku (šablona **rehab-answer.html**)
	/rehab-answer/{id}/delete	DELETE	Smaže odpověď
	/rehab-answer/{id}/move-up	PUT	Změní pořadí odpovědi o jeden stupeň výše
	/rehab-answer/{id}/move-down	PUT	Změní pořadí odpovědi o jeden stupeň níže
	/rehab-answer/{id}/rename	PUT	Přepíše obsah odpovědi
RehabApiController	/rehab-app	POST	Vytvoří novou nebo se napojí na existující instanci mobilní aplikace pacienta v systému. Vrátí id a kód instance.
	/rehab-app/{id}/active-rehab	GET	Vrátí objekt aktivního rehab. plánu pro zobrazení v mobilní aplikaci pacienta
	/rehab-step/{id}	POST	Odešle do systému informaci o ukončení kroku v rehab. plánu (včetně odpovědi v dotazníku)
RehabDbController	/exercise-db	GET	Zobrazí seznam všech cvičení v databázi (šablona **exercise-db.html**)
VideoController	/video	GET	Zobrazí seznam odkazů na nahraná videa (šablona **exercise-videos.html**)
	/video/{filename}	GET	Zobrazí video dle jeho názvu

Obrázek 4.3. Popis RESTful API webové aplikace.

4.1.5 Prezentační vrstva

Na základě návrhu uživatelského rozhraní popsaného v sekci 3.4.1 bylo vytvořeno 8 obrazovek pro správu cviků a rehabilitačních plánů. K jejich implementaci byl použit šablonovací engine Thymeleaf, který renderuje HTML stránky na straně serveru. Výhodou Thymeleaf je, že jeho šablony vypadají stejně jako obyčené HTML stránky s minimálním specifickou syntaxí.

Thymeleaf[5] poskytuje integraci se Spring frameworkem, která umožňuje mapování šablon na jednotlivé kontrolery (viz sekce 4.1.1). V parametru metody kontroleru, která má vracet HTML stránku, se předává objekt `Model` představující logický model šablony. K tomuto objektu se metodou `addAttribute` přidávají data, se kterými lze poté pracovat uvnitř Thymeleaf šablony. Níže je uveden příklad implementace metody zobrazující seznam kategorií cviků. V této metodě se do modelu přidává abecedně seřazený seznam kategorií, kategorie představující koš a objekt představující novou kategorii. Metoda vrací odkaz na šablonu `exercise-categories.html`.

[5] `https://www.thymeleaf.org/`

```
@GetMapping("/exercise-category")
public String getCategories(Model model) {
    List<ExerciseCategoryDTO> categories = categoryService
            .getAllCategories().stream().filter(e -> !e.isBin())
            .sorted(Comparator.comparing(ExerciseCategory::getName))
            .map(ExerciseCategory::toDTO)
            .collect(Collectors.toList());
    ExerciseCategory new_category = new ExerciseCategory();
    ExerciseCategory bin = categoryService.getBin();
    model.addAttribute("categories", categories);
    model.addAttribute("new_category", new_category);
    model.addAttribute("bin", bin);
    return "exercise-categories";
}
```

V rámci Thymeleaf šablony se používají speciální HTML atributy s předponou `th:`, díky kterým lze do stránky vkládat data z modelu (např. atribut `th:text`), dynamicky vytvářet seznamy elementů (atribut `th:each`) nebo např. zobrazovat elementy na základě vyhodnocení logického výrazu (atrubut `th:if`). Na následující ukázce kódu je zobrazena část HTML šablony `exercise-categories.html`, která zobrazuje seznam seřazených kategorií z modelu vytvořeného v metodě `getCategories` na předchozí ukázce.

```
<table id="questions-table">
    <tbody>
    <tr th:each="category: ${categories}" th:id="${category.id}">
        <td th:text="${category.name}"></td>
        <td th:text="${category.exercises.size()+' exercises'}"></td>
        <td>
            <a th:id="${category.id}" class="del">
                <ion-item>
                    <span>Delete</span>
                    <ion-icon name="trash-outline"></ion-icon>
                </ion-item>
            </a>
        </td>
    </tr>
    <tr class="del-row" th:id="${bin.id}">
        <td th:text="${bin.name}"></td>
        <td th:text="${bin.exercises.size()+' exercises'}"></td>
        <td></td>
    </tr>
    </tbody>
</table>
```

Thymeleaf také usnadňuje práci s HTML formuláři pomocí tzv. *command object*[6]. Jedná se o Java objekt jehož atributy představují jednotlivá pole formuláře. Objekt je do formuláře namapován pomocí atributu `th:object` na elementu `form` a atributů `th:field` na polích formuláře. Při změně hodnot polí ve formuláři se automaticky aktualizují i odpovídající atributy tohoto objektu. Po odeslání formuláře na adresu

[6] `https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html`

definovanou v atributu `th:action` se aktualizovaný *command objekt* předá v parametru příslušné metodě v kontroleru, která z něj dostane potřebná data odeslaná z formuláře. Na následující ukázce kódu je zobrazen formulář pro úpravu údajů ke cvičení a příslušný kontroler přijímající objekt formuláře.

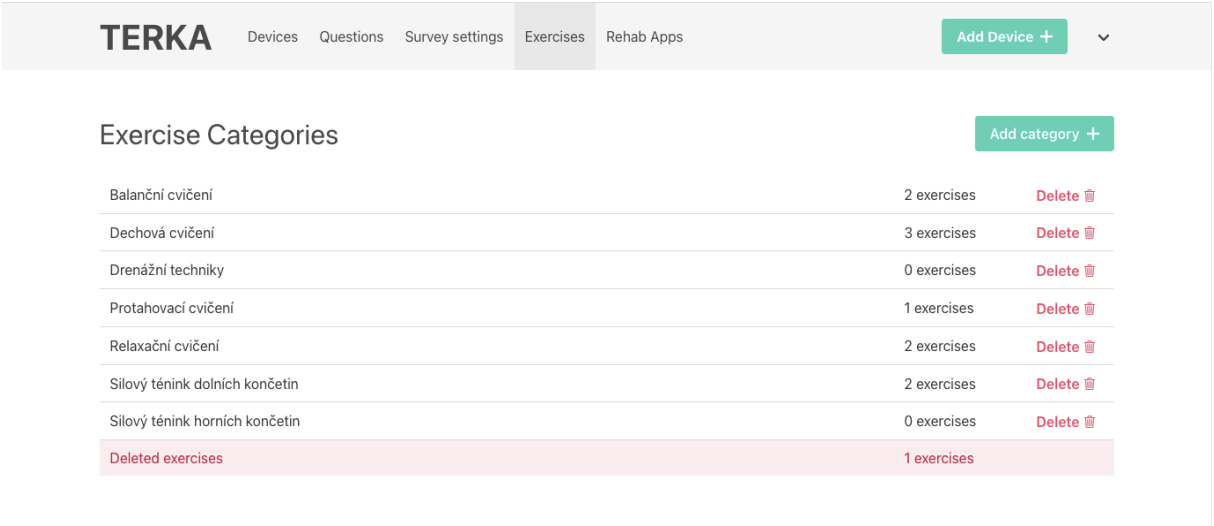
```
<form
    th:action="${'/exercise'}"
    method="post"
    th:object="${exercise}"
>
    <p class="label">Exercise name</p>
    <input class="input" type="text" th:field="*{name}"/>

    <p class="label">Exercise instructions</p>
    <textarea id="text" th:field="*{note}"></textarea>

    <input type="hidden" th:field="*{id}"/>
    <input type="hidden" th:field="*{categoryId}"/>
    <input type="hidden" th:field="*{exerciseRefId}"/>
    ...
</form>
```

```
@PostMapping("/exercise")
public String editExercise(ExerciseDTO exercise) {
    if(exercise.getId() == null) {
        //new exercise
        Exercise e = exerciseService.createExercise(exercise);
        if(e != null) {
            return "redirect:/exercise/" + e.getId();
        }
    }
    ...
}
```

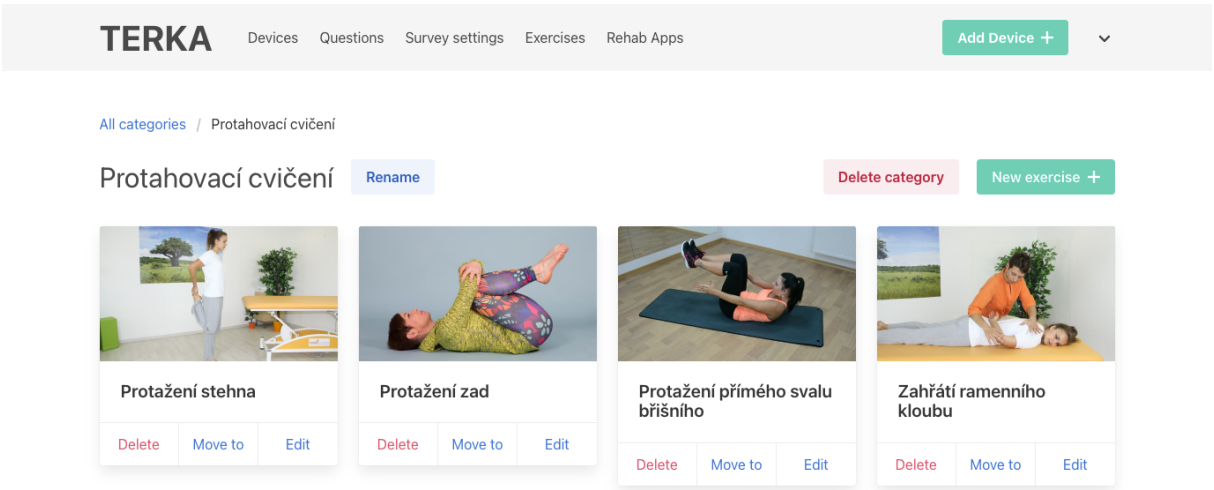
V rámci implementace jednotlivých HTML šablon byla využita CSS knihovna Bulma[7], která poskytuje mnoho předpřipravených stylů pro základní webové prvky jsou odstavce, tlačítka, formulářové prvky nebo tabulky. Dále tato knihovna poskytuje již předpřipravené komponenty, které se běžně vyskytují na webových stránkách jako je např. menu, navigační panely, záložky, karty nebo modální okno. V neposlední řadě Bulma usnadňuje vytváření responzivních stránek díky kontejnerům, které vhodně reagují na změnu rozměrů okna prohlížeče.

Na následujících obrázcích 4.4 až 4.8 jsou zobrazeny ukázky některých výsledných obrazovek webového rozhraní lékaře.

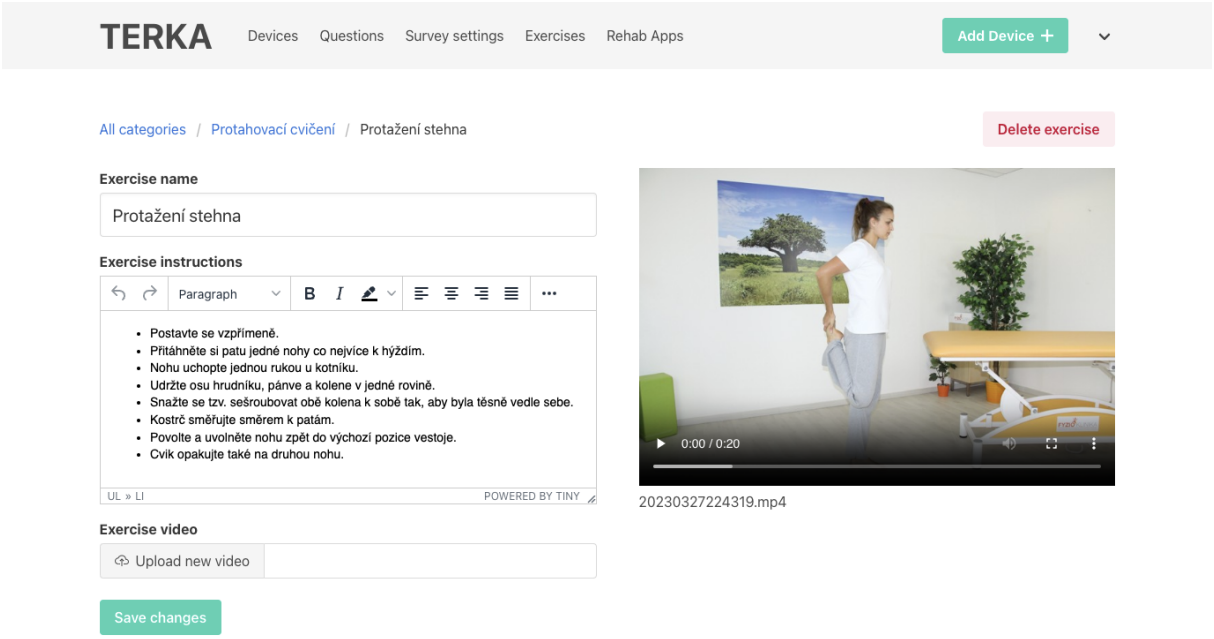
[7] `https://bulma.io/`



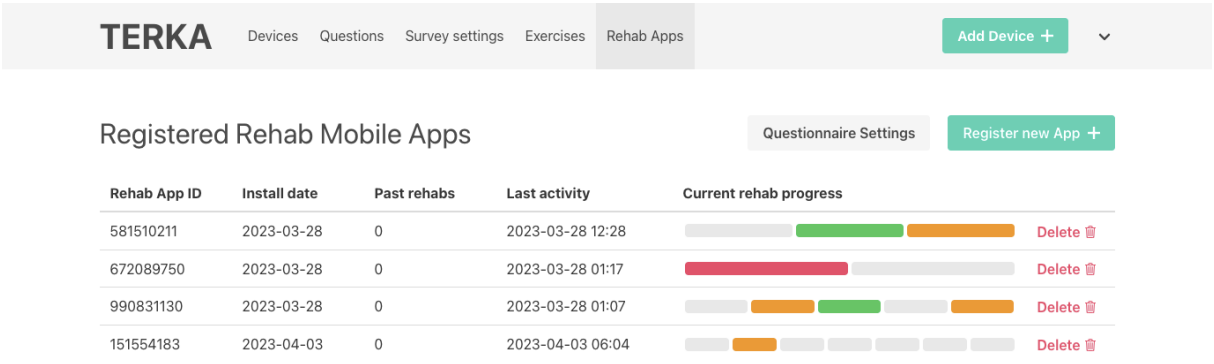
Obrázek 4.4. Obrazovka přehledu kategorií cviků.



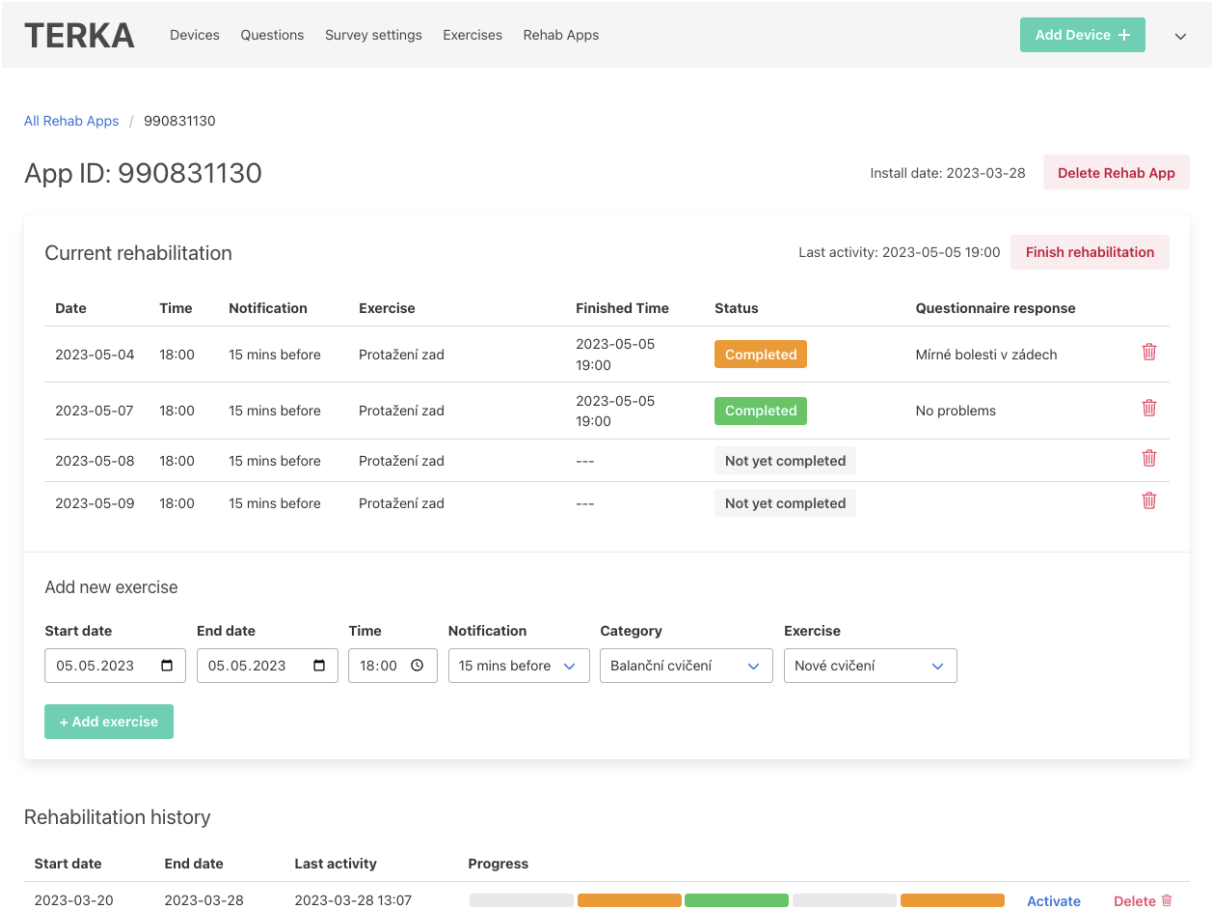
Obrázek 4.5. Obrazovka detailu kategorie cviků.



Obrázek 4.6. Obrazovka detailu cvičení.



Obrázek 4.7. Obrazovka přehledu registrovaných zařízení pacientů.



Obrázek 4.8. Obrazovka zařízení pacienta s aktivním rehabilitačním plánem.

4.2 Implementace mobilní aplikace

Pro implemetaci mobilní aplikace byl využit framework React Native a vývojové prostředí Expo, jejichž použití je popsáno v další sekci. Následuje sekce 4.2.2 vysvětlující implementaci jednotlivých obrazovek a navigace v aplikaci pomocí knihovny React Navigation. V poslední sekci 4.2.3 je pospána komunikace mezi mobilní a webovou aplikací včetně odesílání notifikací pomocí knihovny Expo Notifications.

4.2.1 React Native a Expo

Jak již bylo zmíněno v sekci 3.3.2, framewrok React Native je založený na populární JavaScriptové knihovně React. Klíčovým konceptem této knihovny jsou React komponenty[8]. Komponenta je určitá část uživatelského rozhraní reprezentovaná JavaScriptovou funkcí nebo třídou, která udržuje její stav a vrací kompozici dalších komponent. Uživatelské rozhraní je tak definováno deklarativně podobně jako u statických webových stránek. K tomu se využívá jazyka JSX (JavaScript XML), který rozšiřuje syntaxi JavaScriptu o struktury připomínající HTML dokument. Na následující úkázce je zobrazena třída komponenty představující obrazovku Nastavení sestávající se ze vstupního pole (komponenta `TextInput`), tlačítka (komponenta `TouchableOpacity`) a textu (komponenta `Text`).

[8] `https://react.dev/reference/react/Component`

```
export default function Settings({ navigation }) {
  const [code, setCode] = useState("");

  useEffect(() => {
    getFromStorage("code").then((c) => setCode(c));
  }, []);

  const handleSubmit = async () => {
    removeFromStorage("code");
    removeFromStorage("id").then(() => navigation.replace("Loading"))
  };

  return (
    <View style={styles.inside}>
      <Text style={styles.text}>Kód uživatele</Text>
      <TextInput
        style={styles.textInput}
        value={code}
        showSoftInputOnFocus={false}
      />
      <TouchableOpacity onPress={handleSubmit} style={styles.button}>
        <Text style={styles.buttonText}>Odhlásit se</Text>
      </TouchableOpacity>
      <Text style={styles.text2}>
        Verze aplikace: {Constants.manifest.version}
      </Text>
    </View>
  );
}
```

Jednotlivé komponenty vykreslují UI na základě předaných parametrů a vnitřního stavu. React nese své jméno kvůli způsobu, jakým vykresluje jednotlivé komponenty. Namísto překreslování celého absraktního stromu komponent překresluje React pouze ty komponenty, kterým se změnil stav (tedy *reaguje* na změnu stavu). Díky tomu poskytuje dynamické UI, které rychle reaguje na interakci uživatele. Zároveň je zdrojový kód pro vývojáře snadno čitelný a udržitelný.

Zatímco knihovna React slouží pouze pro tvorbu webových uživatelských rozhraní, React Native tuto knihovnu využívá pro vytváření i mobilních UI pomocí vlastních komponent. V Reactu jednotlivé komponenty odpovídají HTML elementům v DOM (Document Object Model) webové stránky. Framework React Native však využívá nově definované komponenty jako jsou View, Text nebo Button, pomocí kterých vytváří nativní UI pro iOS, Android i webové aplikace.

React Native dále poskytuje podporu pro přístup k nativním funkcím obou platforem. Na oficiálních stránkách frameworku lze nalézt přehled modulů poskytujících API pro přístup k základním funkcím jako je např. modul Appearance[9] pro zjištění tmavého/světlého režimu zařízení nebo Dimensions[10] pro zjištění rozměrů obrazovky. Kromě těchto oficiálních modulů existuje celá řada knihoven aktivně vyvíjených ko-

[9] https://reactnative.dev/docs/appearance
[10] https://reactnative.dev/docs/dimensions

munitou vývojářů jako je např. knihovna React Navigation, která byla použita v této práci (viz 4.2.2).

Expo[11] je sada nástrojů a služeb zjednodušujících proces vývoje React Native aplikací. Expo poskytuje velké množství funkcí a služeb, které vývojářům pomáhají efektivněji vytvářet, nasazovat a publikovat React Native projekty. Vývojové prostředí Expo zahrnuje CLI (command-line interface), které obchází nutnost využití tradičních vývojových prostředí Android Studia a Xcode. Díky tomuto prostředí lze vytvářet strukturované projekty pouze z modulů v jazyce JavaScript, které odstiňují vývojáře od nativního kódu v jazycích Java/Kotlin nebo Objective-C/Swift.

Expo navíc poskytuje vlastní SDK[12] (Software Development Kit) zahrnující další užitečná API a komponenty, jako je přístup ke kameře zařízení, správa notifikací, přístup k senzorům a geolokaci a další. Pro usnadnění testování vyvíjené aplikace na reálném zařízení nebo simulátoru se využívá aplikace Expo Go[13], která je dostupná na obou platformách. Díky funkci *hot reload* může vývojář pomocí této aplikace okamžitě vidět výsledek provedených změn v kódu. Pomocí CLI lze také jednoduše sestavit vyvíjený projekt do binárních souborů .apk pro Android a .ipa pro iOS a publikovat na obchodech aplikací.

4.2.2 Obrazovky a navigace

Na základě návrhu UI mobilní aplikace (sekce 3.4.2) bylo implementováno následujících 5 obrazovek: Registrace, Plán rehabilitace, Cvičení, Dotazník a Nastavení. Pro každou obrazovku byla vytvořena vlastní React komponenta udržující její stav a definující její UI.

React poskytuje tzv. *hooks*, což jsou funkce umožňující správu stavu a dalších vlastností komponent psaných ve funkčním stylu (tzn. jsou definované JavaScriptovou funkcí nikoliv třídou). Mezi tyto funkce patří např. `useState` poskytující přístup ke stavu komponenty nebo `useEffect` umožňující provádět vedlejší efekty (například načítání dat) [23].

Pomocí těchto hook funkcí byla implementována logika jednotlivých obrazovek zahrnující načtení dat do stavu komponenty při otevření obrazovky, vykreslení těchto dat v UI a vyřízení vstupů od uživatele. V ukázce komponenty `Settings` v předchozí sekci je vidět využití funkce `useState` pro přístup k stavové proměnné `code`, jejíž hodnota je vykreslena v UI komponentě `TextInput`. Pomocí funkce `useEffect` je do této proměnné načtena hodnota z objektu v úložišti zařízení.

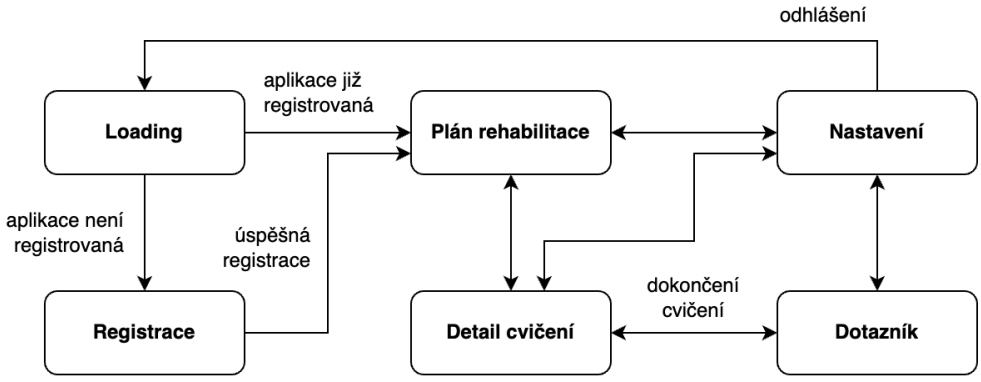
Pro navigaci mezi obrazovkami byla použita knihovna React Navigation[14]. Tato knihovna nabízí několik navigačních stylů jako jsou Stack Navigation, Drawer Navigation nebo Tab Navigation. V rámci mobilní aplikace byla využita zásobníková navigace (Stack Navigation), která ukládá každou novou obrazovku na vrchol zásobníku.

Obrazovka definovaná pomocí Stack Navigation má v horní části panel s názvem obrazovky a ve výchozím nastavení poskytuje možnost vrátit se k předešlé obrazovce v zásobníku. Struktura obrazovek se definuje pomocí komponent NavigationContainer, Stack.Navigator a Stack.Screen. Pomocí Stack.Screen se definuje název a React komponenta implementující UI dané obrazovky. Na následující ukázce je zobrazena část kódu kořenové komponenty aplikace implementující Stack Navigation.

[11] `https://expo.dev/`
[12] `https://docs.expo.dev/versions/latest/`
[13] `https://docs.expo.dev/get-started/expo-go/`
[14] `https://reactnavigation.org/`

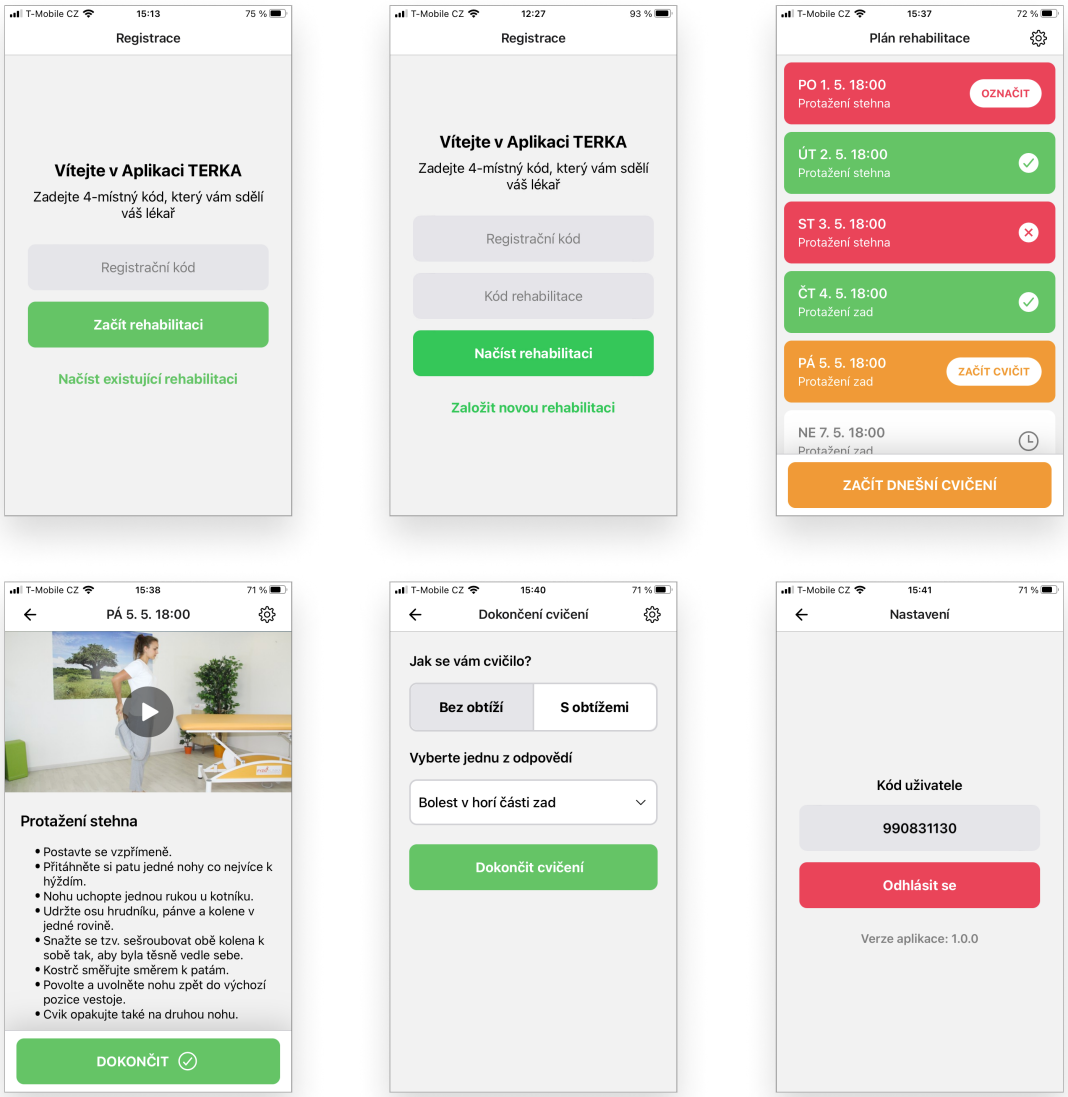
```
export default function Main() {
    ...
    const Stack = createNativeStackNavigator();
    return (
        <NavigationContainer>
            <Stack.Navigator
                initialRouteName="Loading"
                screenOptions={() => ({
                  headerBackVisible: false,
                  headerTitleAlign: "center",
                })}
            >
                <Stack.Screen
                  name={"Loading"}
                  component={Loading}
                  options={{ headerShown: false }}
                />
                <Stack.Screen
                  name={"Login"}
                  component={Login}
                  options={() => ({
                    title: "Registrace",
                  })}
                />
                ...
            </Stack.Navigator>
        </NavigationContainer>
    );
}
```

Kromě pěti zmíněných obrazovek do navigačního procesu přibyla úvodní obrazovka `Loading`, ve které aplikace nejprve zjistí, zda je již zaregistrovaná v systému lékaře. Pokud ano, načte se obrazovka Plánu rehabilitace, pokud ne, zobrazí se obrazovka Registrace. Celý proces navigace v aplikaci je zobrazen na diagramu 4.9.



Obrázek 4.9. Diagram navigace mezi obrazovkami v mobilní aplikaci pacienta.

Výsledná podoba jednotlivých obrazovek je vidět na obrázku 4.10.



Obrázek 4.10. Ukázka obrazovek mobilní aplikace pacienta.

4.2.3 Komunikace s webovou aplikací a správa notifikací

Jak bylo popsáno v sekci 4.1.4, v rámci webové aplikace bylo implementováno RESTful API poskytující mimo jiné 3 rozhraní v rámci kontroleru `RehabApiController` pro komunikaci s mobilní aplikací pomocí JSON objektů. Jejich využití je popsáno níže:

- `POST /rehab-app` - Na tuto adresu jsou odesílána data z obrazovky Registrace (tedy registrační kód a případně kód rehabilitačního plánu). Pokud jsou odeslané údaje správné, webová aplikace registruje mobilní zařízení a vrátí jeho unikátní kód, který je využit pro budoucí komunikaci.
- `GET /rehab-app/{id}/active-rehab` - Z této adresy se načítají jednotlivá cvičení pro obrazovku Plán rehabilitace. Parametr `id` odpovídá identifikátoru mobilního zařízení.
- `POST /rehab-step/{id}` - Na tuto adresu se odesílají informace o dokončení cvičení z obrazovky Dotazník. V rámci zprávy se uvádí čas a druh dokončení (pře-

skočení, dokončení s obtížemi/bez obtíží) včetně případné odpovědi z dotazníku. Příklad JSON objektu této zprávy je uveden níže.

```
{
    "status": 2, //dokončeno s obtížemi
    "message": "Mírné bolesti v zádech",
    "finishedTime": "2023-05-05 19:00"
}
```

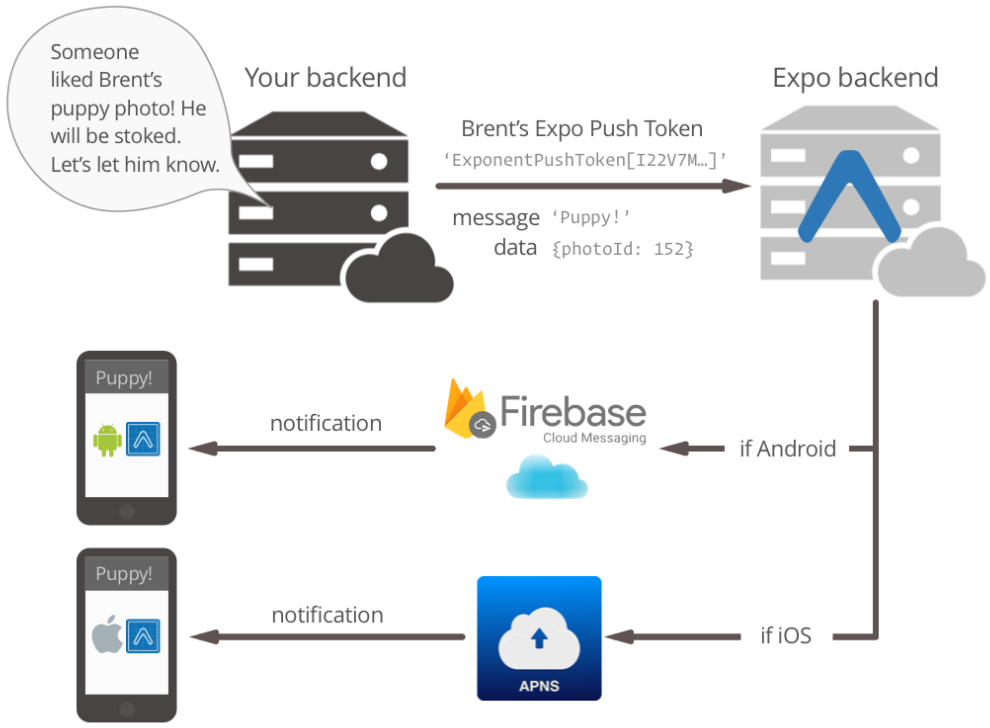
Pro odesílání HTTP požadavků na server byla použita JavaScriptová asynchronní funkce `fetch`, v rámci které se definuje cílová adresa, HTTP metoda, hlavičky a tělo požadavku. Příklad využití této funkce k odeslání informace o dokončení cvičení je uveden na následující ukázce funkce `updateStep`. Parametr `body` představuje JavaScriptový objekt odpovídající JSON objektu uvedeném na předchozí ukázce.

```
const STEP = (id) => DOMAIN + "/api/rehab-step/" + id;

const updateStep = async (stepId, body) => {
  const res = await fetch(STEP(stepId), {
    method: "POST",
    headers: {
      Authorization: "Basic "+Base64.encode(USERNAME+":"+PASSWORD),
      "Content-Type": "application/json",
    },
    body: JSON.stringify(body),
  })
    .catch((e) => null)
    .then((res) => (res ? res.json() : null));
  return res;
};
```

Podobným způsobem byly implementovány i další dvě funkce pro komunikaci s webovou aplikací.

Pro odesílání notifikací do mobilních zařízení byla využita služba Expo Notifications [24], pomocí které lze v React Native aplikaci generovat unikátní token pro budoucí identifikaci příjemce. V rámci aplikace je tento token generován při registraci zařízení a spolu s registračním kódem odeslán do webové aplikace. Při úspěšné registraci je poté uložen v modelu `RehabApp` v atributu `notifToken` (vize skece 4.1.2).

Pomocí Springové anotace `@Scheduled` [25] byla ve webové aplikaci implementována nová služba, která každých 5 minut kontroluje, zda se nějakému mobilnímu zařízení neblíží čas dalšího cvičení v rámci aktuálního rehabilitačního plánu. Pokud je správný čas na odeslání notifikace ohledně blížícího se cvičení, z modelu příslušného zařízení se načte Expo Push Token a společně se zprávou notifikace je odeslán v POST požadavku na Expo server na adrese `https://exp.host/--/api/v2/push/send`. Expo server na základě požadavku poté rozesílá notifikace pomocí služeb Firebase Cloud Messaging (pro Android) a Apple Push Notification service (pro iOS). Schéma procesu je znázorněno na obrázku 4.11.



Obrázek 4.11. Schéma procesu zasílání notifikací pomocí Expo serveru, převzato z [24].

Kapitola 5
Testování

Tato kapitola se zabývá popisem testování vyvíjeného modulu ve třech úrovních. Nejprve jsou jednotlivě otestovány klíčové servisní komponenty ve Spring Boot aplikaci pomocí knihoven JUnit a Mockito. Následně je funkčnost celé webové aplikace otestována přes uživatelské rozhraní automatickým testovacím nástrojem Cypress. V poslední sekci jsou popsány uživatelské akceptační testy.

5.1 Unit testy

Unit testy představují ověřování správného chování nejmenších možných komponent systému (fuknce, metody, třídy). Tyto komponenty jsou testovány izolovaně od ostatních pomocí automatických testů, které často vytváří sami vývojáři pomocí testovacích frameworků. Každý unit test obsahuje vstupní parametry a očekávaný výstup, který je porovnán se skutečným výstupem [26].

Výhodou unit testů je zmíněná izolace testovaných komponent, která zaručuje, že funkčnost komponenty nebude ovlivněna z vnějšku. Díky unit testům lze tak velmi brzo v procesu vývoje odhalit chyby v základních komponentách systému. Zároveň se také předchází vzniku nových chyb při dalším rozšiřování systému.

Pro implementaci unit testů ve webové aplikaci byl použit Spring Boot modul pring-boot-starter-test, který zahrnuje testovací knihovny JUnit a Mockito. JUnit[1] je oblíbená knihovna pro vytváření standardizovaných automatických testů v Javě. Poskytuje anotace `@Test`, `@Before`, `@After` a další, které ozančují metody představující testovací scénáře a přípravné metody, které se spustí na záčátku nebo na konci testování. JUnit také umožňuje ověřovat očekávané výstupy testovaných komponent se skutečnými pomocí metod `AssertEquals`, `AssertTrue` a dalších.

V rámci Spring Boot aplikace byly vytvořeny unit testy pro všechny hlavní metody servisních tříd (viz sekce 4.1.3), které obsahují veškerou logiku aplikace. K zajištění izolace těchto metod během testování od dalších komponent (např. repozitářů) byla použita knihovna Mockito[2]. Tato knihovna poskytuje vytváření tzv. *mocků*, tedy objektů, které mimikují reálné komponenty, ale jejichž chování může být v rámci testovacího scénáře předefinováno programátorem. Pomocí anotací `@Mock` a `@InjectMocks` se definují mockované objekty a metody, ve kterých budou využity. Dále pomocí metod jako jsou `when()` a `thenReturn()` programátor může definovat očekávané chování při zavolání konkrétní metody na mockovaném objektu.

Na následující ukázce je uvedena metoda `moveUp()`, testující stejnojmennou metodu v servisní třídě `RehabSurveyAnswerService`, která umožňuje změnit pořadí odpovědi v dotazníku o úroveň výš. Tato metoda využívá repozitář pro přístup k databázi, jejíž metoda `findById()` je v rámci testu mockována a databáze je reprezentovaná seznamem objektů `ArrayList`.

[1] `https://junit.org/junit5/`
[2] `https://site.mockito.org/`

```
@Mock
RehabSurveyAnswerRepository repository;
@InjectMocks
private RehabSurveyAnswerServiceImpl service;
private List<RehabSurveyAnswer> answers;

@BeforeEach
public void setup() {
    answers = new ArrayList<>();
    lenient().when(repository.findById(anyLong()))
        .then((InvocationOnMock invocation) -> {
            Optional<RehabSurveyAnswer> res = Optional.empty();
            for(RehabSurveyAnswer r : answers) {
                if (r.getId().equals(invocation.getArgument(0))) {
                    res = Optional.of(r);
                    return res;
                }
            }
            return res;
        });
}

@Test
public void moveUp() {
    //prepare
    RehabSurveyAnswer a = service.createAnswer("first");
    RehabSurveyAnswer b = service.createAnswer("second");
    int b_order_before = b.getOrder();
    //when
    service.moveUp(b.getId());
    //then
    assertEquals(1, b_order_before);
    assertEquals(0, b.getOrder());
}
```

5.2 E2E testování

End-to-end (E2E) testování se na rozdíl od unit testů zaměřuje na testování systému jako celku „od začátku do konce“. Testy tohoto typu ověřují, že všechny komponenty systému spolu správně komunikují a dohromady naplňují očekávané požadavky systému v reálném scénáři použití. E2E testování se dělí na vertikální a horizontální. Vertikální testování zahrnuje vytváření testů ve všech vrstvách aplikace od unit testů po testování UI. Horizontální testování ověřuje, zda se z pohledu uživatele systém chová dle očekávání. Tyto testy definují průchod uživatelským rozhraním aplikace a ověřují, zda se uživatel dokáže v systému navigovat [27].

V rámci horizontálního E2E testování byl využit JavaScriptový framework Cypress[3] pro automatické testování webového UI. Tento framework poskytuje funkce pro interakci s testovanou aplikací (vložení textu, stisk klávesy, kliknutí na

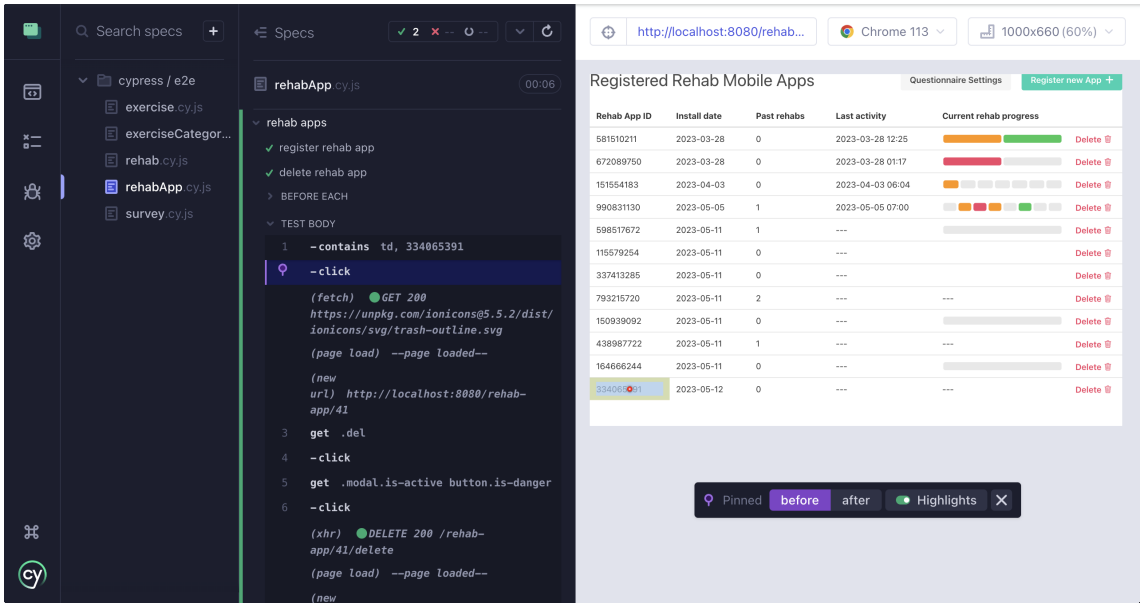
[3] https://www.cypress.io/

odkaz,...) a ověření její zpětné vazby (získání obsahu textového elementu, ověření počtu zobrazených elementů,...). Mezi další funkce tohoto frameworku patří i vytváření a odposlouchávání HTTP požadavků. Díky tomu lze imitovat i interakci jiného systému s testovanou aplikací a ověřovat chování UI.

Ve webové aplikaci byly otestovány podstránky pro kategorie, cvičení, rehabilitační plány, zařízení pacienta a dotazník. Na následující ukázce je zobrazen test smazání aplikace pacienta v systému.

```
it("delete rehab app", () => {
    cy.contains("td", app_code).click();
    cy.get(".del").click();
    cy.get(".modal.is-active button.is-danger").click();
    //assert
    cy.contains("td", app_code).should("have.length", 0);
});
```

Cypress navíc poskytuje vlastní webové rozhraní pro simulaci testů a zobrazení jejich kroků v čase. Tester se tak může „vrátit v čase“ a zobrazit stav aplikace po určitém příkazu v testu. Na obrázku 5.1 je zachyceno vykonávání příkazu `.click()` ze 3. řádku na předchozí ukázce kódu.



Obrázek 5.1. Webové rozhraní testovacího nástroje Cypress.

5.3 Uživatelské akceptační testy

Poslední částí testování jsou uživatelské akceptační testy (UAT). Tyto testy jsou prováděny přímo koncovými uživately pro ověření toho, zda systém splňuje jejich očekávání a je vhodný pro použití v produkčním prostředí. Typicky jsou tyto testy prováděny buď uživately nebo stakeholdery systému. Během testů se ověřují požadavky na systém pomocí případu užití z analytické fáze vývoje (viz kapitola 2.3).

Během psaní této práce k testování systému větším počtem uživatelů bohužel ještě nedošlo.

Kapitola 6
Závěr

Cílem této práce bylo rozšíření stávajícího systému projektu TERESA o modul umožňující telerehabilitaci pacientů. Modul se sestával z webového rozhraní lékaře a mobilní aplikace pacienta. Práce popisovala proces sběru a dokumentace požadavků na tento modul, jeho návrh a následnou implementaci. V závěru práce bylo očekávané chování vyvíjeného systému otestováno atomatickými testy.

Během sběru a dokumentace požadavků bylo identifikováno zadání pro vývoj dvou nových podstránek ve webovém rozhraní lékaře. První měla poskytovat správu cviků reprezentovaných instruktážními videi a texty. Cviky byly pro lékaře uspořádány do kategorií. Druhá podstránka se zabývala tvorbou rehabilitačních plánů pro pacienty. To zahrnovalo propojení mobilní aplikace pacienta se systémem, správu jednotlivých cvičení v rehabilitačních plánech a správu dotazníku po ukončení cvičení.

V rámci práce byly srovnány populární frameworky pro vývoj webových aplikací a posléze byla detailněji popsána struktura frameworku Spring Boot, ve kterém byla aplikace vyvíjena. Následoval popis implementace jednotlivých vrstev aplikace od modelů po RESTful API a prezentační vrstvu implementovanou v šablonovacím enginu Thymeleaf. Práce zahrnovala vysvětlení komunikace mezi front-endem a back-endem při odesílání formulářů v rámci úprav cviků a rehabilitačních plánů na obou podstránkách.

Druhou částí práce tvořil popis vývoje mobilní aplikace pacienta. Během sběru požadavků byl identifikován způsob propojení mobilní aplikace s webovým rozhraním lékaře pomocí registračního kódu. Tento proces byl posléze implementován pomocí HTTP komunikace s RESTful rozhraním. Během návrhu aplikace byly popsány výhody a nevýhody nativního a hybridního přístupu k vývoji mobilních aplikací. Podobně jako u webového rozhraní došlo i v této části k srovnání 3 populárních hybridních frameworků: React Native, Ionic a Flutter. Na základě předchozích zkušeností autora byl pro vývoj zvolen framework React Native. Následoval popis struktury tohoto frameworku s ukázkami kódu z implementace aplikace. Vývoj byl usnadněn nástroji vývojového prostředí Expo, které mimo jiné poskytovalo přístup ke správě notifikací v aplikaci. V závěru implementace bylo popsáno odesílání těchto notifikací pomocí služby v rámci frameworku Spring.

Poslední část práce se věnovala testování jednotlivých komponent ve webové aplikaci pomocí unit testů a poté testování aplikace jako celku pomocí automatických end-to-end testů. Všechny servisní třídy v aplikaci byly pokryty unit testy za použití knihoven JUnit a Mockito. Dále bylo vytvořeno kolem 20ti end-to-end testů pomocí frameworku Cypress, díky kterým byly otestovány všechny hlavní funkce systému přes jeho uživatelské rozhraní.

Dalším krokem ve vývoji bude otestování systému (jak mobilní tak webové aplikace) větším počtem uživatelů (nejlépe lékaři a pacienty), které by ověřilo jeho využití během reálné zdravotní péče.

Literatura

[1] ČVUT. *Projekt TERESA umožní rehabilitaci pacientů po COVID-19 v domácím prostředí*. 2021. `https://intranet.fel.cvut.cz/cz/aktuality/2021/teresa`. [online] (navštíveno dne 2023-05-01).

[2] Miroslav Bures, Katerina Neumannova, Pavel Blazek, Matej Klima, Hynek Schvach, Jiri Nema, Michal Kopecky, Jan Dygryn a Vladimir Koblizek. *A Sensor Network Utilizing Consumer Wearables for Telerehabilitation of Post-acute COVID-19 Patients*. 2022.

[3] Seron P, Oliveros MJ, Gutierrez-Arias R, Fuentes-Aspe R, Torres-Castro RC, Merino-Osorio C, Nahuelhual P, Inostroza J, Jalil Y, Solano R, Marzuca-Nassr GN, Aguilera-Eguía R, Lavados-Romo P, Soto-Rodríguez FJ, Sabelle C, Villarroel-Silva G, Gomolán P, Huaiquilaf S a Sanchez P. *Effectiveness of Telerehabilitation in Physical Therapy: A Rapid Overview*. 2021.

[4] Strubbia C, Levack WMM, Grainger R, Takahashi K a Tomori K. *Use of technology in supporting goal setting in rehabilitation for adults: a scoping review*. 2020.

[5] Alessandro Peretti, Francesco Amenta, Seyed Khosrow Tayebati, Giulio Nittari a Syed Sarosh Mahdi. *Telerehabilitation: Review of the State-of-the-Art and Areas of Application*. 2017.

[6] Omar Mubin, Fady Alnajjar, Nalini Jishtu, Belal Alsinglawi a Abdullah Al Mahmud. *Exoskeletons With Virtual Reality, Augmented Reality, and Gamification for Stroke Patients' Rehabilitation: Systematic Review*. 2019.

[7] David Hailey, Risto Roine, Arto Ohinmaa a Liz Dennett. *Evidence of benefit from telerehabilitation in routine care: a systematic review*. 2011.

[8] Luis Suso-Martí, Roy La Touche, Aida Herranz-Gómez, Santiago Angulo-Díaz-Parreño, Alba Paris-Alemany a Ferran Cuenca-Martínez. *Effectiveness of Telerehabilitation in Physical Therapist Practice: An Umbrella and Mapping Review With Meta–Meta-Analysis*. 2021.

[9] Ralph R. Young. *The Requirements Engineering Handbook*. 2004.

[10] Gopalkrishna Waja, Jill Shah a Pankti Nanavati. AGILE SOFTWARE DEVELOPMENT. *International Journal of Engineering Applied Sciences and Technology*. 2021, 5

[11] J. Reschke. *The 'Basic' HTTP Authentication Scheme*. 2015. `https://datatracker.ietf.org/doc/html/rfc7617`.

[12] Len Bass, Paul Clements a Rick Kazman. *Software Architecture in Practice*. 2012.

[13] C. Richardson. *Microservices Patterns: With examples in Java*. 2018.

[14] *Most Popular Backend Frameworks – 2012/2023*.
https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2023/. [online] (navštíveno dne 2023-05-01).

[15] *MVC*.
https://developer.mozilla.org/en-US/docs/Glossary/MVC. [online] (navštíveno dne 2023-05-01).

[16] *SQL Injection*.
https://www.w3schools.com/sql/sql_injection.asp. [online] (navštíveno dne 2023-05-01).

[17] *Cross-site scripting*.
https://portswigger.net/web-security/cross-site-scripting. [online] (navštíveno dne 2023-05-01).

[18] *Intro to Inversion of Control and Dependency Injection with Spring*.
https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring. [online] (navštíveno dne 2023-05-01).

[19] *Definition: hybrid application (hybrid app)*.
https://www.techtarget.com/searchsoftwarequality/definition/hybrid-application-hybrid-app. [online] (navštíveno dne 2023-05-01).

[20] *Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021*.
https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/. [online] (navštíveno dne 2023-05-01).

[21] *What is a REST API?*
https://www.redhat.com/en/topics/api/what-is-a-rest-api. [online] (navštíveno dne 2023-05-01).

[22] *Introduction to the Java Persistence API*.
https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html. [online] (navštíveno dne 2023-05-01).

[23] *Built-in React Hooks*.
https://react.dev/reference/react. [online] (navštíveno dne 2023-05-01).

[24] *Sending Notifications with Expo's Push API*.
https://docs.expo.io/push-notifications/sending-notifications/. [online] (navštíveno dne 2023-05-01).

[25] *The @Scheduled Annotation in Spring*.
https://www.baeldung.com/spring-scheduled-tasks. [online] (navštíveno dne 2023-05-01).

[26] V. Khorikov. *Unit Testing Principles, Practices, and Patterns*. 2020.

[27] *How to Perform End-to-End Testing*.
https://smartbear.com/learn/automated-testing/how-to-perform-end-to-end-testing/. [online] (navštíveno dne 2023-05-01).

Příloha A
Seznam zkratek

API	Application Programming Interface
CLI	Command-Line Interface
CRUD	Create, Read, Update, Delete
HTTP(S)	Hypertext Transfer Protocol (Secure)
JSON	JavaScript Object Notation
REST	Representational State Transfer
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	eXtensible Markup Language

Příloha B
Obsah elektronické přílohy

`web-app/src/`	zdrojový kód nového modulu ve webové aplikaci (neobsahuje celou webovou aplikaci systému TERESA)
`web-app/cypress/`	zdrojový kód Cypress testů pro webovou aplikaci
`mobile-app/`	zdrojový kód mobilní aplikace