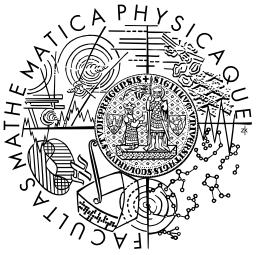


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Patrik Trefil

Platforma pro monitorování mentálního zdraví

Katedra softwarového inženýrství (204. • 32-KSI)

Vedoucí bakalářské práce: Mgr. Petr Škoda, Ph.D.
Studijní program: Informatika (B0613A140006)
Studijní obor: IPP2 (0613RA1400060006)

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Chtěl bych vyjádřit svou vděčnost mé přítelkyni za její podporu a trpělivost, mojí rodině a přátelům za jejich povzbuzení a cenné rady v průběhu mého studia. Velké díky patří také mému vedoucímu práce Mgr. Petru Škodovi, Ph.D., jehož odborné vedení a cenné připomínky byly pro mě velkým přínosem. Díky vám všem za to, že jste byli součástí této cesty.

Název práce: Platforma pro monitorování mentálního zdraví

Autor: Patrik Trefil

Katedra softwarového inženýrství (204. • 32-KSI): Katedra softwarového inženýrství (204. • 32-KSI)

Vedoucí bakalářské práce: Mgr. Petr Škoda, Ph.D., Katedra softwarového inženýrství (204. • 32-KSI)

Abstrakt: V Národním ústavu duševního zdraví vznikla potřeba pro digitalizaci výzkumné a terapeutické praxe. Jedním z procesů k digitalizaci byla spolupráce terapeutů s pacienty/klienty formou dotazníků. V rámci práce byla provedena analýza, návrh a implementace webové aplikace digitalizující tento proces. Aplikace umožňuje tvorbu a zpracování dotazníků. Tuto funkcionalitu lze použít i pro sběr dat pro výzkumné účely. Mezi hlavní kvalitativní požadavky ústavu patřila snadná rozšiřitelnost aplikace o další formy spolupráce a možnost provozu aplikace na vlastní infrastruktuře. Tyto požadavky byly adresovány vhodným rozdělením aplikace na komponenty a využití Docker pro snadné nasazení. Aplikace byla úspěšně předána ústavu k nasazení.

Klíčová slova: vývoj software, webová aplikace, formuláře

Title: Mental health monitoring platform

Author: Patrik Trefil

Department of Software Engineering: Department of Software Engineering

Supervisor: Mgr. Petr Škoda, Ph.D., Department of Software Engineering

Abstract: The National Institute of Mental Health needed to digitalize its research and therapeutic practice. One of the processes was the collaboration of therapists with patients/clients through questionnaires. In this work, we conducted analysis, design, and implementation of a web application digitalizing this process. The application enables the creation and processing of questionnaires. This functionality can also be used for data collection for research purposes. Among the primary qualitative requirements of the institute were the extensibility of the application to include other forms of collaboration and the ability to run the application on its own infrastructure. These requirements were addressed by appropriately dividing the application into components and using Docker for easy deployment. The application was successfully handed over to the institute for deployment.

Keywords: software development, web application, forms

Obsah

Úvod **3**

1 Analýza požadavků **4**
1.1 Stakeholdeři a jejich zájmy . . . 4
1.2 Specifikace . . . 5
1.2.1 Funkční požadavky . . . 6
1.2.2 Nefunkční požadavky . . . 7
1.2.3 Nasazení . . . 8
1.2.4 Monitoring . . . 8
1.2.5 Změny specifikace . . . 8
1.3 Doménový model . . . 8
1.3.1 Omezení . . . 10
1.4 User stories . . . 11
1.4.1 Volitelné . . . 13

2 Analýza existujících řešení **14**
2.1 Problematika správy formulářů . . . 14

3 Návrh aplikace **18**
3.1 Návrh řešení . . . 18
3.2 Architektura . . . 18
3.2.1 Kontext . . . 18
3.2.2 Kontejnery . . . 19
3.2.3 Komponenty . . . 21
3.3 Nasazení . . . 22
3.4 Návrh uživatelského rozhraní . . . 24
3.5 Výběr technologií . . . 24
3.5.1 Software pro práci s formuláři . . . 24
3.5.2 Frontend aplikace . . . 24
3.5.3 Middleware . . . 26
3.5.4 Backend aplikace . . . 27
3.6 Ukládání dat . . . 27
3.6.1 Logický datový model . . . 28
3.6.2 Správa dat o nedokončených vyplnění formulářů . . . 28

4 Vývojová dokumentace **30**
4.1 Nástroje pro vývoj . . . 30
4.2 Soukromí uživatelů . . . 31
4.2.1 GDPR . . . 32
4.3 Autentizace . . . 32
4.4 Konfigurace a modifikace Form.io . . . 33
4.4.1 Modifikace . . . 33
4.4.2 Konfigurace . . . 34
4.5 Propojení správy úkolů a správy formulářů . . . 35

5 Testování aplikace **39**

6 Administrátorská příručka **40**
6.1 Instalace a konfigurace 40
6.2 Spuštění aplikace 41
6.3 Tvorba uživatelských účtů 42
6.3.1 Tvorba účtu pomocí webového rozhraní 42
6.3.2 Tvorba účtu pomocí shell skriptu 42
6.4 Správa aplikace 42

7 Uživatelská dokumentace **43**
7.1 Přihlášení 43
7.2 Užívání aplikace z pohledu zaměstnance 43
7.2.1 Správa formulářů 44
7.2.2 Správa plnitelů 47
7.2.3 Správa úkolů 48
7.2.4 Náhled na odevzdání formuláře 50
7.2.5 Správa účtů 52
7.3 Užívání aplikace z pohledu plnitele 53

8 Zhodnocení vývoje **55**
8.1 Zhodnocení architektury 55
8.2 Zkušenost s Form.io 55
8.3 Zkušenost s NextJS 55
8.4 Open-source vývoj 56
8.5 Spolupráce s NUDZ 57

Závěr **58**

Seznam použité literatury **59**

Seznam obrázků **61**

A Přílohy **62**
A.1 Vstupy pro analýzu požadavků 62
A.2 Wireframy uživatelského rozhraní 63

Úvod

Tato práce vznikla z motivace zlepšit kvalitu poskytované zdravotní péče v oblasti duševního zdraví za pomoci digitálních technologií. Léčba duševních potíží se nazývá psychoterapie a trvá obvykle několik měsíců až let. V průběhu terapie pacient pravidelně dochází na sezení s terapeutem. Na těchto sezeních terapeut zadává pacientovi různé úkoly, které pacient následně vypracuje mimo sezení. Úkoly mají často formu dotazníků, mentálních cvičení nebo vzdělávacích aktivit jako je čtení článků. Tyto úkoly jsou často zadávány papírovou formou a následně ručně vyhodnocovány. Digitalizací procesu spolupráce lze zajistit efektivnější spolupráci mezi terapeuty a pacienty a automatizovaný sběr dat o pacientech. Získaná data lze využít pro sledování vývoje stavu pacientů a také pro výzkum. Cílem je tedy posílení spolupráce mezi terapeuty a jejich pacienty/klienty.

Výzkum z oblasti psychologie může mít mnoho forem. Jedna z častých forem výzkumu je sběr dat o skupině pomocí dotazníků. Dotazníky mají často papírovou formu a následně jsou ručně vyhodnocovány. Digitalizací sběru dat lze zajistit efektivnější sběr dat a automatizované vyhodnocování. Vzhledem k požadavkům této domény na digitální nástroje jako je nutnost zachování anonymity pacientů/-klientů/účastníků studie, není možné použití obecných nástrojů jako je Google Forms. Psychoterapie a výzkum v oblasti psychologie jsou oblasti, které mohou získat velký prospěch z digitalizace, ale vyžadují specializované nástroje.

Tato práce byla vytvořena ve spolupráci s Národním ústavem duševního zdraví (NUDZ), který poskytl svou expertízu a zkušenosti v oblasti duševního zdraví. Tato instituce se zabývá výzkumem neurobiologických a psychosociálních mechanismů spojených se vznikem a průběhem nejzávažnějších duševních poruch [9]. Ústav současně poskytuje i psychiatrickou péči nemocným a plánuje řešení používat.

Iniciativa ke spolupráci s NUDZ přišla z mé strany a byla přijata s otevřeností. V rámci spolupráce došlo k plné realizaci procesu vývoje softwaru od sběru požadavků, přes analýzu, design až po implementaci, testování a předání. Vývoj byl v agilním duchu proložen schůzkami, kde byly prezentovány dosažené výsledky a byla vedena diskuze ohledně dalších kroků.

Nyní popíšeme strukturu práce. Na začátku provedeme analýzu požadavků Národního ústavu duševního zdraví a popíšeme doménu (kapitola 1). Následně prozkoumáme již existující software, který v práci použijeme (kapitola 2). Ná základě průzkumu navrhneme řešení (kapitola 3). Poté popíšeme jakým způsobem jsme postupovali při implementaci a proč jsme tento postup zvolili (kapitola 4). V další kapitole se zaměříme na testování aplikace (kapitola 5). Vysvětlíme, jak se aplikace provozuje (kapitola 6) a jak se aplikace užívá z pohledu terapeta/-výzkumníka a pacienta/klienta/účastníka studie (kapitola 7). Dále zhodnotíme jednotlivé části procesu vývoje a zhodnotíme celkovou úspěšnost projektu (kapitola Zhodnocení vývoje). Na závěr shrneme dosažené výsledky a navrhneme možnosti dalšího vývoje (kapitola 8.5).

1. Analýza požadavků

Pro analýzu požadavků proběhlo několik online schůzek s pracovníky Národního ústavu duševního zdraví. Na úvodní schůzce se probíraly možnosti využití výpočetních technologií v oblasti duševního zdraví. Na této schůzce jsme se shodli na tvorbě platformy pro digitalizaci práce s dotazníky. Na následujících schůzkách byly zadavateli kladeny otevřené otázky týkající se představ o fungování aplikace. Také byly získávány informace o stakeholderech, omezeních a IT infrastruktuře zadavatele. Podařilo se také získat ukázkový dotazník, který by měla aplikace zvládnout zpracovat. Ukázkový dotazník lze najít v příloze A.1.

V této kapitole se první podíváme na stakeholdery a jejich zájmy, což je popsáno v sekci 1.1. Výsledkem úvodních schůzek byla specifikace požadavků, která je v sekci 1.2. První dvě sekce mají neformální charakter a jsou zformalizovány v sekci 1.4. Tato konverze byla provedena zejména pro lepší organizaci vývoje softwaru. Doména se ukázala býti poměrně složitá a bylo důležité detailně zdokumentovat její fungování. Dalším krokem bylo tedy vytvoření doménového modelu, který dostal formu diagramu a textového souboru definic jednotlivých entit. Tato dokumentace nám poskytla společný jazyk pro další komunikaci a je zachycena v sekci 1.3. V průběhu tvorby modelu došlo k ujasnění některých požadavků a fungování domény.

1.1 Stakeholdeři a jejich zájmy

Tato sekce popisuje stakeholdery a jejich zájmy, které byly identifikovány během analýzy požadavků. Stakeholdeři jsou rozděleni do následujících skupin: uživatelé aplikace, vývojáři, provozovatel software, vedení organizace. Skupina uživatelů aplikace je rozdělena i na podskupiny pro zachycení většího detailu. Uživatelské role jsou podrobněji definovány a popsány dále v sekci 1.2.

Uživatelé aplikace

- Podpora mobilních zařízení.

Plnitelé dotazníků

- Jednoduché rozhraní pro plnění úkolů, které zajistí zlepšení jejich zdravotního stavu a poskytne terapeutům potřebné informace pro jejich léčbu.

Terapeuti

- Jednoduché rozhraní vhodné pro uživatele s nízkými technickými znalostmi.
- Jednoduchý přístup k výsledkům pacientů s přehlednou vizuální reprezentací.
- Spolupráce s plniteli v rámci terapie formou domácích úkolů pro zlepšení poskytované péče a prevenci kriminality.

Výzkumníci

- Možnost tvorby komplexních výzkumných studií.
- Možnost pokročilé analýzy dat vlastním nástrojem.

Vývojáři

- Dobře dokumentovaný kód a vhodně dokumentovaná architektura aplikace.
- Dobře nastavené procesy pro prevenci chyb jako jsou testy, kontinuální integrace, apod.
- Jednoduchá rozšiřitelnost aplikace.
- Vysoká modifikovatelnost aplikace.

Provozovatel software

- Jednoduché nasazení aplikace na server.
- Schopnost monitorování aplikace.

Vedení organizace

- Nízká cena za vývoj a provoz aplikace.
- Nízké časové nároky na zaučení plnitel.
- Vysoké zabezpečení aplikace.

1.2 Specifikace

Tato sekce obsahuje neformální specifikaci požadavků na aplikaci. Soupis požadavků byl od začátku vývoje organizován do kategorií. Organizace požadavků byla ponechána v původní podobě. Požadavky jsou rozděleny do čtyř kategorií: funkční požadavky, nefunkční požadavky, nasazení a monitoring. Pro účely tohoto dokumenty byly požadavkům přiřazeny identifikátory. Funkční požadavky jsou označeny R-FR-ID, nefunkční požadavky R-NR-ID, požadavky na nasazení R-DR-ID a požadavky na monitoring R-MR-ID, kde ID je číslo požadavku z dané kategorie. Požadavky na monitoring jsou označeny R-MR-ID a požadavky na nasazení R-DR-ID, kde ID je opět číslo požadavku z dané kategorie. Mnoho požadavků pracuje s uživatelskými rolemi, které jsou definovány v požadavku R-NR-1. Specifikace vznikla v rané fázi vývoje a některé požadavky byly později změněny. Změny jsou popsány v sekci 1.2.5.

1.2.1 Funkční požadavky

R-FR-1 Zadavatel/správce dotazníků může zadávat dotazníky a úkoly konkrétnímu plniteli, nastavit frekvenci opakování a nastavit start/deadline.

- Start je čas, kdy lze dotazník nejdříve vyplnit.
- Deadline je čas, kdy lze dotazník nejpozději vyplnit. Deadline může být dvou typů: striktní a uvolněný. V případě nastavení striktního deadline *nelze* po uplynutí deadline dotazník vyplnit. V případě nastavení uvolněného deadline *lze* dotazník vyplnit i po uplnutí deadline.
- Dotazník se skládá z otázek, které mohou přijímat odpovědi těchto typů:
  - text
  - více možností (možno zvolit právě jeden)
  - více možností (možno zvolit libovolný počet)
- U každé otázky je možno nastavit podmíněné zobrazení. Jinak řečeno, otázka se plniteli zobrazí, pokud je splněna podmínka definovaná správcem dotazníku při vytvoření.

R-FR-2 Správce dotazníků je schopen vytvářet/mazat dotazníky a způsoby vyhodnocení

R-FR-3 Správce dotazníků může v dotazníku upravit obsah otázek a možných odpovědí v případě, že se jedná o volbu z možností. *Nelze* přidat/odebrat otázky, přidat/odebrat možné odpovědi, ani upravit způsob výpočtu skóre.

- Při tvorbě dotazníku je možno nastavit automaticky počítané metriky pomocí vzorce (např. `otázka1 + otázka2 - otázka4`, kde `otázka1`, `otázka2` a `otázka4` jsou proměnné reprezentující výsledky příslušných otázek).

R-FR-4 Plnitel může v aplikaci vypracovat dotazníky a úkoly, které mu byly přiděleny správcem/zadavatelem dotazníků.

R-FR-5 Plnitel je schopen vyplnit část dotazníku, uložit si dosud zodpovězené otázky a v budoucnu vyplňování dotazníku dokončit i na jiném zařízení.

R-FR-6 Plnitel může smazat svá data.

R-FR-7 Správce/zadavatel dotazníků je schopen vidět výsledky dotazníků a úkolů všech plnitelů.

R-FR-8 Správce/zadavatel může vybrat data, která budou vizualizována na grafech.

R-FR-9 Správce dotazníků/člen technické podpory je schopen vytvářet/mazat účty plnitelů, účty pro ostatní správce a účty pro zadavatele dotazníků.

R-FR-10 Správce dotazníků/člen technické podpory je schopen měnit přístupová práva všech ostatních účtů.

R-FR-11 Plintel je schopen měnit heslo svého účtu a je schopen svůj účet smazat.

R-FR-12 Výsledky dotazníků plnitelů může správce/zadavatel dotazníků z aplikace exportovat do formátu CSV.

R-FR-13 Aplikace bude pouze v Českém jazyce, ale bude připravena na internacionalizaci.

R-FR-14 Plnitel v systému vystupuje pod ID, které je náhodně vygenerováno.

Viditelnost vyhodnocení dotazníků

R-FR-15 Plnitel je schopen vidět vyhodnocení svých dotazníků, u kterých to správce povolil.

R-FR-16 Správce dotazníků může dovolit plniteli vidět výsledky konkrétního dotazníku - číselná hodnota/graf/text.

R-FR-17 Viditelnost výsledků lze nastavit následujícími způsoby:

- plnitel vidí výsledky
- plnitel nevidí výsledky
- plnitel vidí výsledky pouze, pokud výsledek dotazníku splňuje podmínku (např. výsledné skóre je větší než 10)

Volitelné

Požadavky z této podsekce nejsou kritické pro fungování aplikace, ale mohou zjednodušit užívání aplikace.

R-FR-18 Sledování délky času stráveného nad jednotlivými otázkami, času, kdy plnitel dotazník vyplnil, počtu změn odpovědí.

R-FR-19 E-mail upozornění na nové úkoly, na nesplněné úkoly.

R-FR-20 Push notifikace v prohlížeči na nové úkoly, na nesplněné úkoly.

1.2.2 Nefunkční požadavky

R-NR-1 V systému vystupují uživatelé s následujícími rolemi:

- Plnitel = pacient/klient
- Zaměstnanci NUDZ
  - Správce dotazníků = terapeut/výzkumník
  - Zadavatel dotazníků = terapeut/výzkumník
- Technická podpora

R-NR-2 Uživatelské rozhraní by mělo být vhodné pro uživatele s nízkými technickými znalostmi.

R-NR-3 K programu by měla být dodána uživatelská a technická dokumentace.

R-NR-4 Aplikace bude maximalizovat použití již existujícího software, pro snížení ceny, časové náročnosti a zjednodušení následné údržby.

1.2.3 Nasazení

R-DR-1 Aplikace bude spouštěna v kontejneru.

R-DR-2 Aplikace poběží na serveru s následujícími parametry: (lze domluvit navýšení v případě potřeby)

- CPU: 2 jádra
- RAM: 4 GB
- HDD: volných 29 GB

1.2.4 Monitoring

R-MR-1 Bude k dispozici rozhraní pro monitorování aplikace.

1.2.5 Změny specifikace

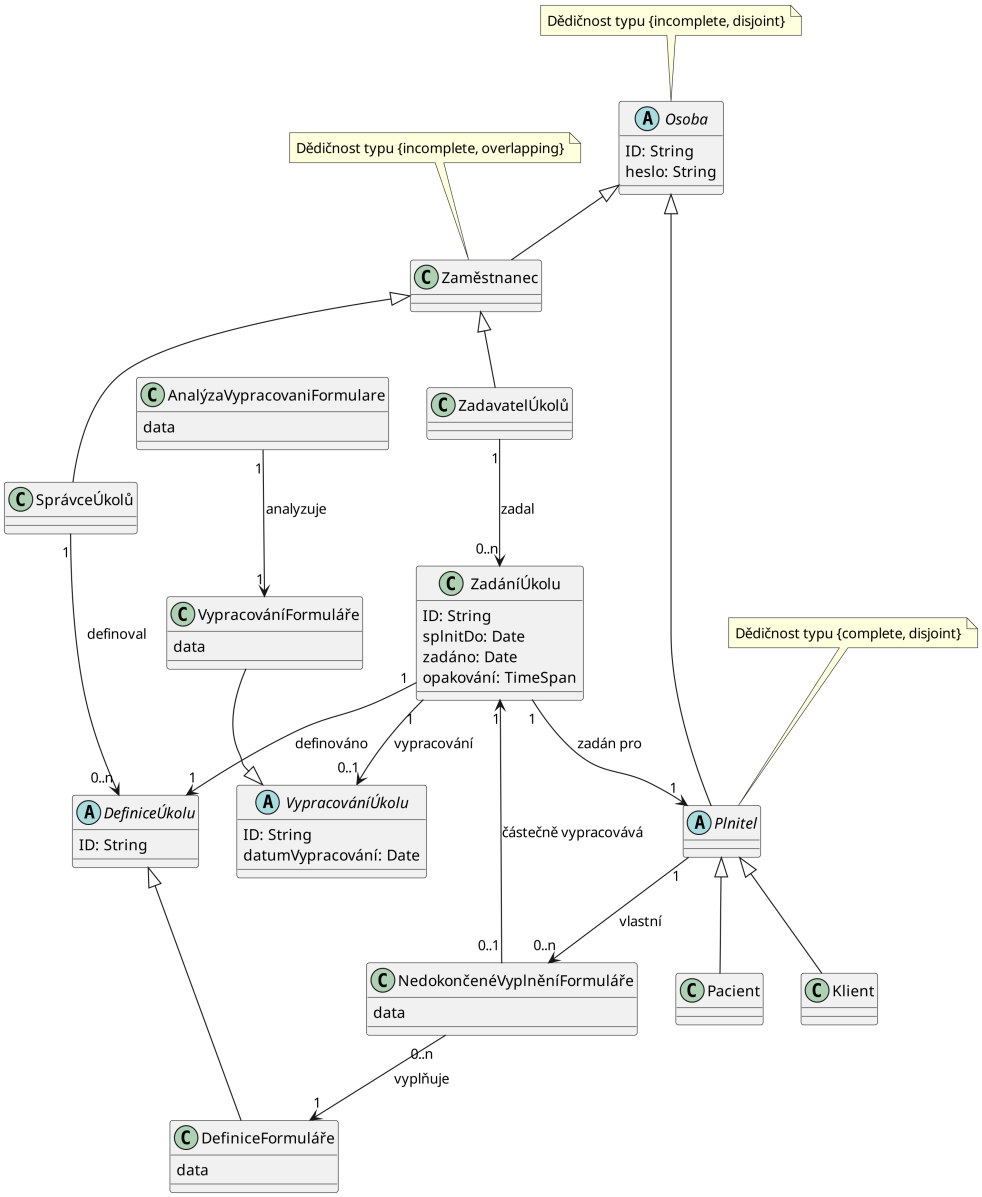
V průběhu vývoj se ukázalo, že některé požadavky popsané v sekci 1.2.1 je potřeba pozměnit. Tato podsekce popisuje všechny provedené změny.

Původní specifikace pracovala s možností nastavení viditelnosti vyhodnocení dotazníků. Toto bylo pokryto požadavky R-FR-15, R-FR-16 a R-FR-17. Možnost s nastavením viditelnosti na základě splnění podmínky byla zavrhnuta z následujícího důvodu. Pokud by plnitel neviděl vyhodnocení dotazníku, tak by si mohl odvodit, že výsledek dotazníku je nepříznivý. Toto zjištění by mohlo mít negativní vliv na jeho zdravotní stav. Možnost nastavit vyhodnocení dotazníku jako vždy viditelné pro plnitele bylo zavrhnuto ze stejného důvodu. Vyhodnocení složitějších dotazníků navíc může správně interpretovat pouze odborník.

Původně bylo zamýšleno, že si plnitelé budou přímo schopni smazat svá data. Vzhledem k tomu, že některá data budou určena pro výzkumné účely, což může vyžadovat nutnost archivace, tak není vhodné umožnit plnitelům přímo smazat svá data. Bylo tedy rozhodnuto, že plnitelé budou moci o smazaní svých dat pouze požádat skrze pracovníka NUDZ. Každý požadavek bude pak posouzen individuálně.

1.3 Doménový model

Na základě schůzek a specifikace ze sekce 1.2 jsme vytvořili doménový model. Model byl následně konzultován s pracovníky NUDZ a byl schválen. Model je popsán diagramem na obrázku 1.1. Jednotlivé entity jsou definovány níže. Omezení kladené na model, která nejsou vyjádřena v diagramu, jsou popsána v sekci 1.3.1.



Obrázek 1.1: Doménový model formou Class diagramu v notaci UML[1]

Nyní přesněji definujeme jednotlivé entity vystupující v diagramu:

Osoba Lidská bytost, která má vztah k systému.

Zaměstnanec Osoba, která je zaměstnána v NUDZ.

SprávceÚkolů Zaměstnanec, který je zodpovědný za správu úkolů v systému.

ZadavatelÚkolů Zaměstnanec, který zadává pro plnitelé úkoly v systému.

Plnitel Osoba, která vypracovává úkoly v systému.

[1]Dědičnost v UML vyjadřuje jiný vztah než v programování. Třída v UML je množina a odvozená třída je její podmnožina. Vztahy mezi odvozenými třídami, tedy podmnožinami, mohou být 4 typů podle toho zda-li se podmnožiny překrývají a zda-li je jejich sjednocení množina definovaná třídou, od které odvozujeme. Tento vztah je vždy popsán v poznámce k třídě, od které odvozujeme [18].

Klient Plnitel, který je klientem NUDZ (platí za poskytovanou péči).

Pacient Plnitel, který je pacientem NUDZ (poskytovaná péče je hrazena pojišťovnou).

DefiniceÚkol Obecný potenciálně znovupoužitelný popis činnosti plnitele.

DefiniceFormuláře Definice úkolu, která obsahuje formulář, který je určen k vyplnění plnitelem.

ZadáníÚkolu Zadání úkolu pro konkrétního plnitele na základě definice úkolu.

VypracováníÚkolu Vypracování úkolu plnitelem na základě zadání úkolu.

VypracováníFormuláře Vypracování úkolu ve tvaru vyplnění formuláře.

AnalýzaVypracováníFormuláře Odvozená data z vypracování formuláře, například výsledné skóre.

NedokončenéVyplněníFormuláře[2] Částečné vyplnění formuláře.

1.3.1 Omezení

Zde jsou vypsána omezení, která nejsou vyjádřena v diagramu. Každé omezení je napsáno jak v neformální textové podobě, tak v Object constraint language.

NedokončenéVyplněníFormuláře

Nedokončené vyplnění formuláře pro definici formuláře, zadání úkolu a plnitele může existovat pouze pokud je zadání úkolu pro plnitele a zadání úkolu zadává stejný formulář, který je částečně vyplněn nedokončeným vyplněním formuláře.

```
context plnitel: Plnitel inv
  plnitel->vlastni->forAll(
      nedokonceneVyplneniFormulare |
          nedokonceneVyplneniFormulare
            ->castecneVypracovava
            ->zadanPro = self
          and
          nedokonceneVyplneniFormulare
            ->castecneVypracovava
            ->definovano = nedokonceneVyplneniFormulare->vyplnuje
          )
```

ZadáníÚkolu

Definice úkolu musí logicky odpovídat vypracování úkolu. Např. nemůžeme považovat přečtení článku jako vypracování úkolu, který je definován jako vypracování formuláře.

[2] Umožňuje plniteli přerušit vyplňování formuláře a pokračovat později.

```
context z: ZadáníÚkolu inv
    if z.vypracování.oclIsKindOf(VypracováníFormuláře) then
        z.definováno.oclIsKidnOf(DefiniceFormuláře)
    endif
```

VypracováníÚkolu

ID musí být unikátní.

```
context v1, v2: VypracováníÚkolu inv v1.ID = v2.ID implies v1 = v2
```

DefiniceÚkolu

ID musí být unikátní.

```
context d1, d2: DefiniceÚkolu inv d1.ID = d2.ID implies d1 = d2
```

Osoba

ID musí být unikátní.

```
context o1, o2: Osoba inv o1.ID = o2.ID implies o1 = o2
```

1.4 User stories

Zde je seznam požadavků ve sjednocené formě. Požadavky mají formu „Jako <role> chci <funkčnost>, abych/protože/aby <cíl>“. Každý požadavek má u sebe i cíl, jelikož může být důležitý pro porozumění požadavku a může ovlivnit způsob implementace. Jedná se o přepis funkčních požadavků z neformální specifikace ze sekce 1.2. Role uživatelů jsou definovány v nefunkčním požadavku R-NR-1 ve specifikaci. Tato sekce pracuje již s změnami specifikace, které byly popsány v sekci 1.2.5. Každá položka má identifikátor ve formátu R-US-ID, kde ID je číslo položky.

R-US-1 Jako zaměstnanec chci mít možnost založit účet pro plnitele, abych mohl využít systém v rámci terapie/výzkumu.

R-US-2 Jako plnitel chci, aby moje data nemohli číst ostatní plnitelé, protože jsou soukromá.

R-US-3 Jako terapeut/výzkumník chci, aby byl plnitel schopen vidět/plnit jen úkoly, které mu byly zadány, aby se zachovala integrita sbíraných dat.

R-US-4 Jako plnitel chci být schopen zobrazit svá data, abych věděl co je o mě v systému evidováno (např. vyplněné dotazníky).

R-US-5 Jako správce dotazníků chci mít možnost zadávat dotazníky a úkoly konkrétnímu plniteli, nastavit frekvenci opakování a nastavit start/deadline, abych mohl správně koordinovat procesy a aktivity v systému.

R-US-6 Jako správce dotazníků chci mít možnost vytvářet otázky v dotaznících, které mohou přijímat odpovědi typu text, více možností (možno zvolit právě jeden) a více možností (možno zvolit libovolný počet), abych mohl vytvářet různorodé a komplexní dotazníky.

R-US-7 Jako správce dotazníků chci mít možnost nastavit podmíněné zobrazení otázky, takže otázka se plniteli zobrazí pokud je splněna podmínka definovaná při vytvoření, což umožní flexibilní a přizpůsobené průzkumy.

R-US-8 Jako správce dotazníků chci mít možnost vytvářet, mazat dotazníky a způsoby vyhodnocení, abych mohl udržovat aktuální a relevantní soubor dotazníků.

R-US-9 Jako správce dotazníků chci mít možnost nastavit automaticky počítané metriky pomocí vzorce (např. `otázka1 + otázka2 - otázka4`), abych mohl efektivně vyhodnocovat odpovědi.

R-US-10 Jako plnitel chci mít možnost vypracovat dotazníky a úkoly, které mi byly přiděleny správcem/zadavatelem dotazníků, abych mohl aktivně participovat v systému.

R-US-11 Jako plnitel chci mít možnost vyplnit část dotazníku, uložit si dosud zodpovězené otázky a v budoucnu vyplňování dotazníku dokončit, aby bylo vyplňování dotazníků flexibilní a pohodlné.

R-US-12 Jako plnitel chci mít možnost smazat svá data, abych měl kontrolu nad svými daty.

R-US-13 Jako správce/zadavatel dotazníků chci mít možnost vidět výsledky dotazníků a úkolů všech plnitelů a vybrat data, která budou vizualizována na grafech, abych mohl monitorovat a analyzovat výsledky.

R-US-14 Jako správce dotazníků/člen technické podpory chci mít možnost vytvářet/mazat účty plnitelů, účty pro ostatní správce a účty pro zadavatele dotazníků, abych mohl efektivně spravovat uživatelské účty.

R-US-15 Jako správce/zadavatel dotazníků chci mít možnost exportovat výsledky dotazníků plnitelů z aplikace do formátu CSV, abych mohl dělat pokročilé analýzy dat.

R-US-16 Jako plnitel chci, aby k programu byla dodána uživatelská a technická dokumentace, abych mohl lépe porozumět, jak aplikace funguje a jak ji používat.

R-US-17 Jako technický pracovník chci, aby aplikace byla spouštěna v kontejneru, abych mohl snadno spravovat a nasazovat aplikaci.

R-US-18 Jako technický pracovník chci mít k dispozici rozhraní pro monitorování aplikace, abych mohl sledovat výkon a stav aplikace a rychle reagovat na potenciální problémy.

R-US-19 Jako plnitel chci v systému vystupovat pod ID, které je náhodně vygenerováno, aby bylo zajištěno mé soukromí.

R-US-20 Jako správce dotazníků chci být schopen upravovat text již existující otázky, abych mohl opravit překlepy nebo vylepšit formulaci otázky.

R-US-21 Jako plnitel chci měnit heslo svého účtu, abych zajistil bezpečnost svého účtu.

1.4.1 Volitelné

Požadavky z této podsekce nejsou kritické pro fungování aplikace, ale mohou zjednodušit užívání aplikace.

R-US-25 Jako plnitel chci dostávat e-mail upozornění na nové úkoly a na nesplněné úkoly, abych byl vždy informován o svých úkolech a termínech.

R-US-26 Jako plnitel chci dostávat push notifikace v prohlížeči na nové úkoly a na nesplněné úkoly, abych byl vždy aktuálně informován o svých úkolech.

R-US-27 Jako výzkumník chci mít možnost sledovat chování uživatele při odpovídání na otázky (např. délka času strávená nad jednotlivými otázkami, čas, kdy uživatel dotazník vyplnil, počet změn odpovědí), abych mohl lépe porozumět výsledkům.

2. Analýza existujících řešení

V době analýzy existujících řešení (od sprna 2022 do září 2022) nebylo nalezeno žádné řešení pro online spolupráci terapeutů a pacientů/klientů. Pro vyhledávání byl použit primárně vyhledávač Google. Také bylo využito vyhledávání na platformách GitHub a GitLab. Pro vyhledání byla použita klíčová slova jako *therapy*, *collaboration*, *software*, *mental health* apod.

Velkou část aplikace, kterou chceme vyrobit, tvoří systém pro správu formulářů. Protože nebylo nalezeno kompletní řešení, v této kapitole se zaměříme na analýzu existujících řešení pro tvorbu formulářů a sběr dat. V kapitole popíšeme nalezené řešení a jejich výhody a nevýhody. Pokud se ukáže, že je některý z řešení vhodný pro naše použití, tak jej začleníme do naší aplikace.

2.1 Problematika správy formulářů

Hledaná řešení lze rozdělit ná několik částí. První část je rozhraní pro tvorbu formulářů. V našem případě by to mělo být drag and drop rozhraní, které umožní vytvářet formuláře bez nutnosti znalosti programování. Druhá část je formát pro ukládání vytvořených formulářů. Při uložení je potřeba ukládat nejenom strukturu formuláře, ale také styly jednotlivých prvků a logiku formuláře. Logika obsahuje například podmíněné zobrazení otázek, přechody mezi stránkami formuláře v případě, že se jedná o vícestrankový formulář, či validaci vstupu. Třetí část je vykreslení uložených formulářů. Tato část na základě uložené definice formuláře vykreslí formulář, který je následně možné vyplnit. Čtvrtá část je sběr dat z formulářů. Sběr dat zahrnuje ukládání jednotlivých odpovědí, metadata o vyplnění formuláře a také zpřístupnění sesbíraných dat terapeutům. Pátá část je možnost uživatele uložit částečně vyplněný formulář a dokončit vyplnění později. Dokončení může proběnout i na jiném zařízení. Tato schopnost systému byla uvedena jako požadavek R-FR-5. Budeme primárně hledat řešení, které řeší všechny tyto části. Pokud se však ukáže, že existuje sada nástrojů, které lze kombinovat a dohromady řeší všechny tyto části, tak je také zvážíme.

Nástroje mohou mít formu služby nebo knihovny. Mezi hlavní požadavky na nástroj pro správu formuláře patří možnost podmíněného zobrazení otázek, kvalitní dokumentace a možnost hostovat na vlastní infrastruktuře. Důležité kritérium pro výběr je samozřejmě i cena. Pro hledání nástrojů byly použit vyhledávač Google, vyhledávání na platformách GitHub a GitLab. Pro vyhledávání byly použity klíčová slova jako *form builder*, *form management*, *form rendering*, *form serialization* apod. Srovnání nástrojů je shrnuto na konci kapitoly v tabulce 2.1.

Google Formuláře

Google Formuláře je jeden z nejrozšířenějších nástrojů pro tvorbu formulářů. Formuláře v tomto nástroji lze snadno vytvářet v prohlížeči a následně sdílet pomocí odkazu s plniteli. Jeden formulář může spravovat více uživatelů. Výsledky formulářů lze analyzovat pomocí nástroje Google Tabulky či exportovat do souboru. Google Formuláře podporují uložení částečně vyplněných formulářů a do-

končení vyplnění později na jiném zařízení, ale pouze pro uživatele, kteří jsou přihlášení do svého Google účtu [13]. Není možné požadovat po uživatelích, aby se přihlásili do svého Google účtu, jelikož sbíraná data musí být asociovaná pouze s náhodně vygenerovanými identifikátory. Pro naše účely tedy Google Formuláře požadavek na uložení částečně vyplněných formulářů nepodporuje. Nevýhodou Google Formulářů je, že neumožňují přizpůsobení vzhledu formulářů. Další nevýhodou je, že Google Formuláře sbírají data o uživatelích, kteří formulář vyplňují. Google Formuláře nenabízejí variantu s hostováním na vlastní infrastruktuře. Produkt je zdarma pro osobní použití, ale jinak stojí ke dni 20. 3. 2023 $12 na uživatele, který formuláře spravuje, měsíčně.

Form.io

Form.io poskytuje veškeré funkce, které jsou vyžadovány zadavatelem. Form.io umožňuje velkou míru upravení vzhledu a fungování formulářů. Form.io je používáno mnoha velkými společnostmi jako je Visa či ICANN. Tento software je v zahraničí využíván i ve státních institucích. Konkrétně se používá pro provoz pohraničních sil Britské vlády [4]. Form.io umožňuje uložit částečně vyplněný formulář a dokončit vyplnění později i na jiném zařízení [12]. Software lze hostovat na vlastní infrastruktuře. Dle ceníku ke dni 20. 3. 2023 varianta s hostováním na vlastní infrastruktuře však stojí $900 měsíčně.

OpnForm

OpnForm je open-source řešení správy formulářů, které je aktivně vyvíjeno. Je to poměrně mladý projekt, který byl založen v září roku 2022. Projekt nemá ke dni 13. 3. 2023 žádnou dokumentaci ani ceník. Tedy nelze posoudit, zda-li splňuje požadavky zadavatele.

Typeform

Typeform je úspěšný nástroj pro tvorbu online formulářů. Nástroj je velmi vyspělý a poskytuje mnoho funkcí včetně větvení formulářů na základě předchozích odpovědí. Typeform nenabízí variantu s hostováním na vlastní infrastruktuře. Tento software také neumožňuje sledování chování uživatele [20]. Typeform sbírá data o uživatelích, kteří formulář vyplňují [19]. Nástroj umožňuje uložit částečně vyplněný formulář a dokončit vyplnění později, ale pouze v rámci jednoho zařízení [14]. V případě potřeby více než 5 uživatelských účtů záleží cena na dohodě.

Knihovny pro práci s formuláři

Pro práci s formuláři existuje mnoho knihoven. Knihovný buď řeší uživatelské rozhraní pro tvorbu formulářů, vykreslování formulářů nebo obě tyto části.

Může se zdát, že by bylo možné použít jednu knihovnu pro tvorbu formulářů a jinou pro vykreslování formulářů. Problémem však je interoperabilita mezi

knihovnami. Způsob ukládání definic formulářů je totiž různý. Najít jednu knihovnu pro tvorbu formulářů a jinou nezávislou knihovnu pro vykreslování formulářů se tedy nepovedlo. Proto se podíváme na knihovny, které řeší oba problémy.

První zvážená knihovna se nazývá uniforms. Tato knihovna řeší vykreslování formulářů a podporuje několik různých formátů definice formulářů. Knihovna navíc není závislá na konkrétním UI frameworku, což je velká výhoda. Tato knihovna je vyvíjena firmou Vazco. Část knihovny, která je určena pro vykreslování formulářů je však součástí placeného produktu. Firma Vazco byla kontaktována ohledně možnosti možné spolupráce s žádostí o bezplatné využití software pro vykreslování formulářů. V odpovědi jsme obdrželi povolení pro bezplatné užití software pro vykreslování formulářů pro tento projekt. Po obdržení potvrzení však firma přestala reagovat na veškeré e-maily. Navzdory několika dalším pokusům o kontaktování firmy dopadlo toto jednání neúspěšně. Část knihovny, která je určena pro vykreslování definic formulářů, je možné použít pouze po individuální dohodě s firmou. Vzhledem k neúspěsnoti v jednání s firmou Vazco toto nebylo možné. Část knihovny pro definici formulářů využívá vlastní formát pro ukládání definic formulářů. Tedy by nebylo možné použít tuto část knihovny společně s jinou knihovnou pro vykreslování formulářů. Z těchto důvodů byla tato knihovna vyřazena z výběru.

Další alternativou je knihovna Formily. Formily je open-source projekt, který má ke dni 20. 3. 2023 přes 9 tisíc hvězdiček v repozitáři na platformě Github. Za projektem navíc stojí velká komerční firma Alibaba. Projekt je aktivně vyvíjen, ale bohužel v době volby technologie nebyla dostupná dokumentace v anglickém jazyce. Z důvodu chybějící dokumentace nebylo možné ověřit, zda-li knihovna splňuje všechny požadavky zadavatele. Proto byla tato knihovna vyřazena z výběru.

Závěr

Nepodařilo se nám najít žádnou vhodnou kombinaci knihoven, která by alespoň částečně řešila náš problém. Vlastnosti nalezených služeb jsou shrnuty v tabulce 2.1. Jak je vidět z tabulky, tak jediným řešením, který splňuje všechny požadavky zadavatele je Form.io. Toto řešení je však poměrně drahé a proto byl zvolen kompromis. Jádro Form.io je open-source a kromě několika málo funkcí je možné jej využít zdarma. Jádro tedy využijeme jako základ našeho řešení a doplníme jej o chybějící funkce. Pokud bude náš software úspěšný a bude potřeba mít k dispozici všechny funkce Form.io, tak je možné dokoupit placenou verzi. Jak konkrétně tento software použijeme v naší aplikaci bude popsáno v sekci 3.5.1.

	Form.io	Google Formuláře	OpnForm	TypeForm
Podmíněné zobrazení	✓ Ano	✓ Ano	✓ Ano	✓ Ano
Kvalitní dokumentace	✓ Ano	✓ Ano	✗ Ne	✓ Ano
Self-hosting	✓ Ano	✗ Ne	✓ Ano	✗ Ne
Nesbírá data o uživatelích	✓ Ano	✗ Ne	Neznámé	✗ Ne
Uložení nedokončeného vyplnění	✓ Ano	✗ Ne	Neznámé	✗ Ne
Cena	$900	$12 na uživatele	$0	Neznámé

Tabulka 2.1: Srovnání existujících řešení

3. Návrh aplikace

Tato kapitola popisuje, jak byla aplikace navržena. Nejprve popíšeme jakým způsobem jsme se rozhodli řešit požadavky zadavatele v sekci 3.1. Poté navrhneme architekturu systému v sekci 3.2. Následně se budeme věnovat návrhu uživatelského rozhraní formou wireframů (sekce 3.4). Následně popíšeme výběr technologií (sekce 3.5) a způsob ukládání dat v systému (sekce 3.6).

3.1 Návrh řešení

Jedním z požadavků byla dostupnost aplikace jak na počítačích, tak na mobilních zařízeních. Zvážili jsme dvě možnosti, jak tohoto dosáhnout. První možností je vyvinout mobilní aplikaci a desktopovou aplikaci zvlášť. Druhou možností je vyvinout webovou aplikaci s responzivním rozhraním, která bude dostupná na všech zařízeních. Jelikož nepotřebujeme žádné nativní funkce zařízení, tak je výhodnější vyvinout pouze jednu webovou aplikaci. Webová aplikace má také výhodu v tom, že ji není potřeba instalovat a aktualizovat. Další výhodou řešení webovou aplikací je, že zadavatel má zkušenosti s jejich provozem a má existující infrastrukturu.

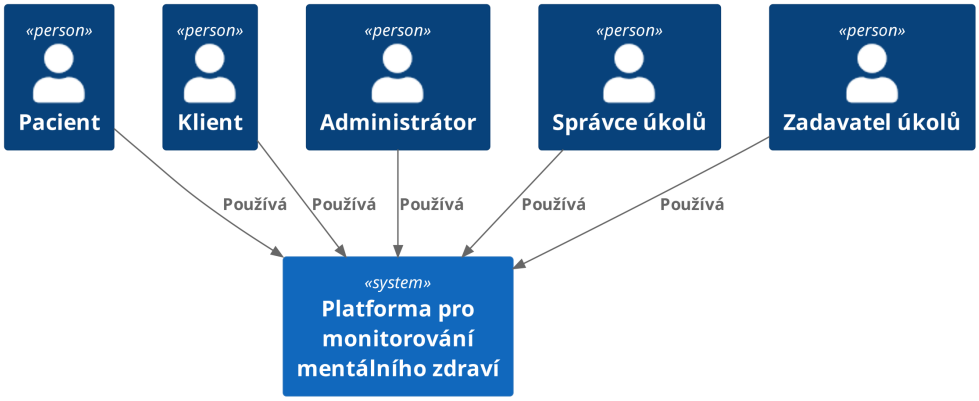
Řešení webovou aplikací vyžaduje vývoj webové stránky a serveru. Nároky na uživatelské rozhraní jsou v dnešní době velmi vysoké. Rozhraní musí být rychlé, responzivní a intuitivní. V případě serveru je potřeba se zaměřit zejména na návrh rozhraní a bezpečnost. Budeme se snažit použít co nejvíce již existujícího software, jak bylo zadáno v požadavku R-NR-4 Naším cílem bude vybrat software, který nám dá dostatek flexibility a zároveň bude dostatečně stabilní a udržitelný.

3.2 Architektura

Nyní navrhneme architekturu aplikace. Budeme využívat architekturu typu klient-server. Klientem bude webová aplikace, která bude spuštěna ve webovém prohlížeči uživatele. Serverová část bude mít distribuovanou architekturu, jelikož bude využívat službu Form.io. Pro popis architektury použijeme C4 model a zaměříme se na 3 úrovně abstrakce - kontext, kontejnery a komponenty.

3.2.1 Kontext

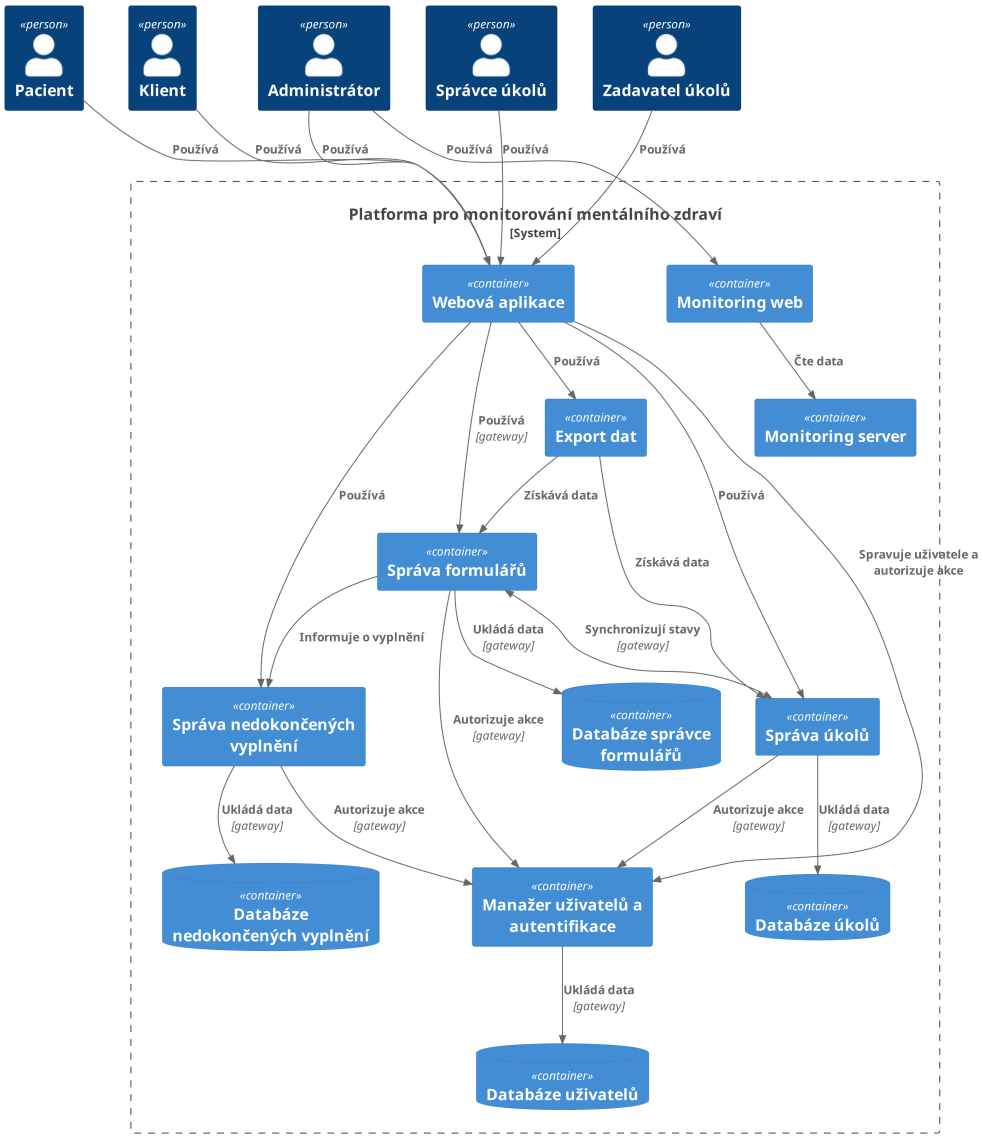
Začneme kontextovou vrstvou (Obr. 3.1), která zobrazuje vztahy mezi naším systémem, jeho uživateli a ostatními systémy.



Obrázek 3.1: C4 model - Systémový kontext

3.2.2 Kontejnery

Nyní popíšeme dekompozici našeho systému na kontejnery a vztahy mezi nimi (Obr. 3.2). Aplikace byla rozdělena na kontejnery primárně na základě logické souvislosti funkcionalit, které poskytují. U většiny kontejnerů je zřejmý jejich účel. Některé však vyžadují bližší popis, který se nachází níže.



Obrázek 3.2: C4 model - Kontejnery

> Pokud je vztah mezi dvěma kontejnery vyznačený jako gateway, znamená to, že komunikace probíhá skrze prostředníka, který poskytuje vrstvu abstrakce. Gateway snižuje provázanost komunikujících systémů (coupling) a zajišťuje modifikovatelnost. Bližší informace o tomto vzoru lze nalézt v článku Gateway od Martin Fowler.

> Monitoring server monitoruje všechny ostatní kontejnery. Relace byly vynechány pro zachování čitelnosti diagramu.

Správa formulářů

Tento kontejner poběží na serveru a bude umožňovat CRUD[1] operace nad definicemi formulářů. Také bude poskytovat rozhraní pro sběr dat z formulářů a jejich export.

Správa úkolů

Tento kontejner poběží na serveru a bude umožňovat CRUD[1] operace nad úkoly.

Správa nedokočených vyplnění

Tento kontejner se bude starat o ukládání nedokončených vyplnění formulářů. Tento kontejner bude ve své podstatě fungovat jako uložiště typu klíč-hodnota, kde klíčem je identifikátor formuláře a hodnotou jsou nedokončená vyplnění. Pro zachování jednoduchosti systému nebude kontejner posílat požadavky na kontejner pro správu formulářů. Kontejner tedy nebude validovat existenci formuláře pro který je nedokončené vyplnění ukládáno ani nebude kontrolovat, zda-li nedokončené vyplnění obsahuje validní data. Přístup ke všem operacím, které tento kontejner poskytuje, dostanou pouze plnitelé úkolů, což je skupina uživatelů definovaná v požadavku R-NR-1. Způsob ukládání dat, které vlastní tento kontejner, je popsán v sekci 3.6.2.

Export dat

Pro export sesbíraných dat z aplikace potřebujeme získat data o úkolech a odevzdání formulářů. Každý úkol spojíme s odevzdáním, které bylo vytvořeno v rámci splnění tohoto úkolu. Data o úkolech a odevzdání však vlastní dvě různé služby. Spojení dat můžeme provést na klientovi a nebo na serveru. Obě varianty jsou možné. Umístění procesu spojení na server zajistí lepší interoperabilitu našeho systému s ostatními systémy a umožní programově řídit export dat z aplikace.

Pro tyto účely byl vytvořen speciální kontejner jehož úkolem je získat data z ostatních služeb a spojit je. Jelikož export dat nebude probíhat příliš často a sesbíraná data budou poměrně malá (očekáváme maximálně řádově stovky záznamů), není nutné dbát na vysokou výkonnost této služby.

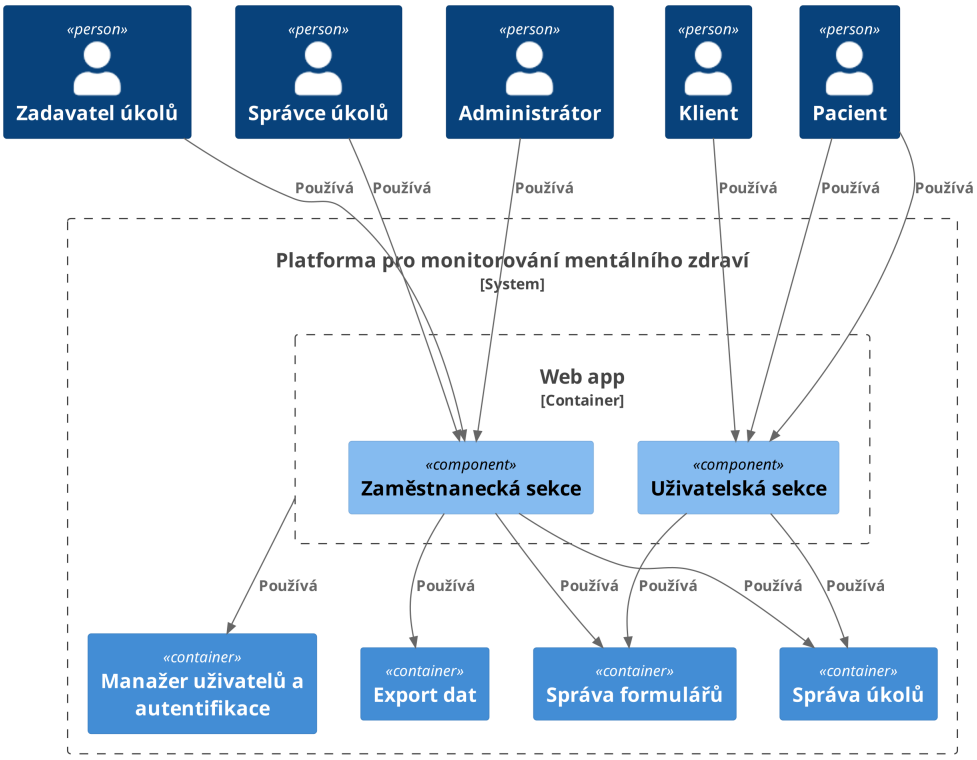
Máme několik možností implementace této služby. První možností je použití návrhového vzoru API composition. Tento vzor pracuje se spojením uvnitř operační paměti. Implementace tohoto vzoru je velmi jednoduchá, ale spojení dat může být neefektivní. Druhou možností je použití návrhového vzoru Command query responsibility segregation. Tento vzor využívá repliky dat z obou datových zdrojů. Tato možnost je efektivnější, ale složitější na implementaci.

Z vlastností obou možností je vidět, že API composition je vhodnější řešení našeho problému. Finální implementace byla realizována vytvoření tRPC routeru v rámci NextJS aplikace (Obr. 3.4).

3.2.3 Komponenty

Dělení na komponenty popíšeme pouze pro kontejner *Webová aplikace*. Tento kontejner rozdělíme na zaměstnaneckou a uživatelskou sekci (Obr. 3.3).

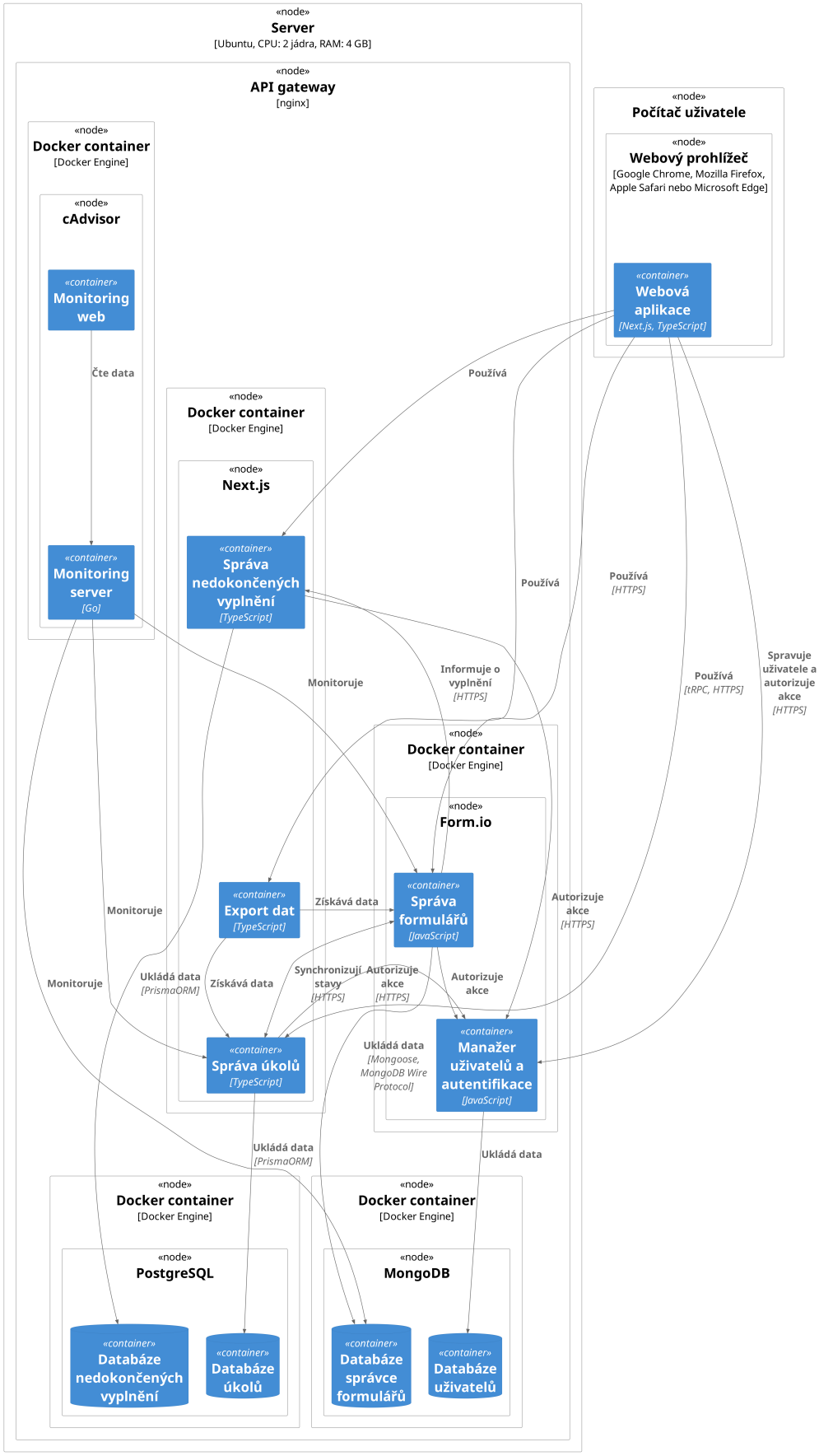
[1]CRUD je zkratka pro *create, read, update, delete*, což jsou základní operace při práci s daty [1]



Obrázek 3.3: C4 model - Komponenta webová aplikace

3.3 Nasazení

Na obr. 3.4 je znázorněn diagram nasazení pomocí notace C4 modelu. Diagram obsahuje i použité technologie. Odůvodnění výběru těchto technologií je popsáno v sekci 3.5. Všechny komponenty jsou nasazeny na jediný server, jak bylo požadováno v zadání v sekci 1.2.3. Pro snížení počtu nasazených Docker kontejnerů byly kontejnery Databáze nedokončených vyplnění a Databáze úkolů nasazeny na jednu instanci databáze. Implementace dovoluje i nasazení těchto komponent na různé instance databáze.



Obrázek 3.4: Diagram nasazení

3.4 Návrh uživatelského rozhraní

Jak již víme z kapitoly 1 o analýze požadavků, budeme potřebovat navrhnout rozhraní pro plnitele úkolů a také pro terapeuty. Předpokládáme, že všichni uživatelé aplikace mají nízké technické dovednosti a proto se budeme snažit navrhnout co nejjednodušší rozhraní. Abychom předešli vývoji rozhraní, které by zadavatel nepovažoval za vhodné, tak vytvoříme wireframy, které necháme zadavatelem schválit. Výsledné wireframy naleznete v přílohách práce A.2.

3.5 Výběr technologií

V této sekci se budeme zabývat výběrem konkrétních technologií, které budeme používat při vývoji aplikace. Chceme volit populární a vyspělé nástroje, jelikož programátoři, kteří budou aplikaci v budoucnu rozšiřovat, je pravděpodobně budou znát. Zároveň také chceme co nejméně komplikovat workflow a nasazení. Budeme vybírat primárně z open-source řešení.

3.5.1 Software pro práci s formuláři

Software pro práci s formuláři je klíčovou součástí aplikace, jak je popsáno v kapitole 2. Problematika správy formulářů je detailně popsána v sekci 2.1. V závěru kapitoly 2 jsme se rozhodli pro využití software Form.io. Nyní popíšeme, jak přesně tento software bude využit.

Jádro software Form.io je open-source a má licenci Open Software License 3.0, která dovoluje komerční použití. Jádro také obsahuje základní systém pro správu uživatelů. Tato bezplatná část má samozřejmě jistá omezení oproti placené verzi. Nevýhodou je, že použití vlastního autentizačního poskytovatele je dostupné pouze v placených verzích. Navzdory omezením bezplatné verze využijeme jádro software Form.io pro správu formulářů. Do budoucna se nabízí přechod na placenou verzi.

V době učinění tohoto rozhodnutí bylo předpokládáno, že funkce pro uložení nedokončených vyplnění je součástí jádra. Ukázalo se, že přestože tato funkce není v dokumentaci označená jako placená, tak je dostupná pouze v placené verzi. Dojmu, že se jedná o bezplatnou funkce, napomáhá i to, že v bezplatném rozhraní pro tvorbu formulářů existují tlačítka pro uložení nedokončených vyplnění. Až poté, co se ji uživatel pokusí použít, zjistí že jsou nefunkční. Z těchto důvodů bylo nutné implementovat vlastní řešení pro uložení nedokončených vyplnění.

3.5.2 Frontend aplikace

Frontend aplikace je klientská část aplikace běžící v prohlížeči. Hlavní problémy této části aplikace pro nás budou rychlost, responzivita a design. V dnešní době existuje mnoho knihoven a frameworků, které řeší tyto problémy. Níže jsou popsány zvolené technologie, knihovny a odůvodnění jejich použití.

Bootstrap Knihovna, která zjednodušuje stylizaci webových aplikací. Jedná se o dlouho existující, vyspělou a populární knihovnu. Alternativy jako Tailwind CSS nabízejí větší míru flexibility, ale neposkytují žádné hotové kom-

ponenty. Tento projekt nemá za cíl vytvářet vlastní designovou identitu, proto je výhodnější použít hotové komponenty, které Bootstrap nabízí.

JavaScript/TypeScript JavaScript je výchozí programovací jazyk pro vývoj webových aplikací. Alternativní řešení jsou v současné době omezená. TypeScript je nadstavba JavaScript poskytující statickou typovou kontrolu, což pomůže předejít mnoha chybám. Přestože TypeScript zkomplikuje a zpomalí vývoj, tak jej použijeme pro prevenci chyb.

React React je knihovna používaná mimo jiné pro vývoj webových aplikací. Tento nástroj zajišťuje reaktivní aktualizaci uživatelského rozhraní na základě změn dat aplikaci. React je dle průzkumu State of JavaScript z roku 2022 nejpoužívanější front-end framework. Jedná se o vyspělý framework, který má velkou komunitu a mnoho existujících knihoven.

NextJS Místo volby jednotlivých balíků pro základy řešení routing, middleware, sdílených layoutů apod. zvolme framework, který nám tyto funkce poskytne. NextJS je meta-framework postavený na nástroji React, který rozšiřuje jeho funkce. Jedná se o full-stack framework, takže jej využijme i na serverovou část.

react-bootstrap Tato knihovna značně zjednodušuje použití knihovny Bootstrap v React aplikacích a navíc zajišťuje alespoň základní webovou přístupnost.

react-i18next Aplikace bude podporovat více jazyků a proto je vhodné použít knihovnu, která nám usnadní práci s překlady. Oproti alternativám jako Polyglot.js nebo Format.js nabízí knihovna react-i18next více funkcí. Knihovna react-i18next je s 2.9 milióny stažení za týden nejpopulárnější ze zvažovaných knihoven dle počtu stažení z npm registry [11].

React formio Knihovna pro vykreslování formulářů na základě schématu. Tato oficiální knihovna je součástí projektu Form.io. Tato knihovna také poskytuje podporu pro autentizaci uživatele pomocí Form.io serveru. Tuto funkci však nebudeme používat z důvodů popsaných v bodě o NextAuth.js.

NextAuth.js Knihovna pro autentizaci uživatelů v prohlížeči i na serveru. Nástroj pracuje skvěle s NextJS a lze jeho konfigurace je relativně jednoduchá. Ukázalo se, že knihovna Formio React není vhodná pro autentizaci uživatele. První důvod je, že vyžaduje, aby si vývojář psal vlastní logiku pro ochranu stránek, přesměrování apod. To zbytečně vytváří prostor pro chyby. Druhým důvodem je, že inicializace autentizace se dělá pouze na straně klienta, jelikož knihovna nepodporuje server-side rendering. To má značně negativní vliv na výkon aplikace. Třetí důvod je špatná dokumentace knihovny. Poslední výhodou použití jiné knihovny než Formio React je vytvoření vrstvy abstrakce v kódu nad autetifikačním systému. Díky tomuto je možné kdykoliv vyměnit autentizační systém za jiný. Jako alternativy jsem zvažoval Passport.js a NextAuth.js. Passport.js je více nízko-úrovňový a opět vyžaduje, aby si vývojář psal vlastní logiku. Oproti Formio React má však pěknou dokumentaci. Nakonec jsem se ale přiklonil k NextAuth.js.

Zod Pro validaci dat na serveru a validaci formulářů na klientovi využijeme validační knihovnu. Knihovnu Zod jsem zvolil jelikož má rozhraní, se kterým se dobře pracuje.

React hook form Aplikace bude obsahovat formuláře pro přihlášení, správu uživatelů, tvorbu úkolů a mnoho dalších. Formuláře lze tvořit pomocí přímé interakce s prvky HTML dokumentu. Rozhraní těchto prvků je však špatně navrženo například proto, že používají pro všechny hodnoty typ `string`. Pro lepší práci s formuláři využijeme knihovnu React hook form. Díky této knihovně budeme moci snadno validovat formuláře, získávat hodnoty formulářů se správnými typy a zobrazovat chybové hlášky. Lze zvážit také knihovnu Formik. Recenze na internetu byly stejně dobré jako pro React hook form, ale integrace s validační knihovnou Zod existuje pouze jako komunitní balíček. React hook form má oficiální integraci s validační knihovnou Zod pomocí balíčku resolvers.

Tanstack query Aplikace bude obsahovat mnoho dat, která budou načítána ze serveru. Pro zvýšení kvality kódu a zlepšení výkonu aplikace využijeme knihovnu. Populární řešení jsou Tanstack query a SWR. Tanstack query oproti SWR nabízí více funkcí a mnoho z nich nám značně ulehčí práci. Mezi tyto funkce patří například vývojové nástroje. Tanstack query má oficiální vývojové nástroje s širokou funkcionalitou. Vývojové nástroje pro SWR existují pouze jako komunitní balíček, který má pouze omezenou funkcionalitu.

Tanstack table Aplikace bude obsahovat velké množství tabulek. Naše tabulky budou podporovat filtrování, řazení a stránkování. Toto jsou běžné funkce, na které již existuje spoustu řešení. Vzhledem k tomu, že používáme knihovnu Tanstack query, tak se nabízí použít i knihovnu Tanstack table. Výhodou této knihovny je, že je velmi flexibilní a umožňuje vytvářet vlastní komponenty pro vykreslování tabulek. Jedná se o tzv. *headless UI* knihovnu, což nám umožní definovat vlastní vzhled tabulek.

Recharts Aplikace bude potřebovat vizualizovat sesbíraná data a proto potřebujeme knihovnu na vykreslování grafů. Pro tyto účely byly zváženy knihovny visx od AirBnB, echarts od Apache a Recharts. Knihovna visx má velmi komplikovanou dokumentaci i příklady. Tento nástroj má evidentně strmou výukovou křivku a pro naše účely je zbytečně komplexní. Knihovna echarts je velmi populární, ale nemá oficiální podporu pro React. Knihovna Recharts má kvalitní dokumentaci obsahující jak jednoduché tak složitější příklady a navíc má přímou podporu pro React.

3.5.3 Middleware

Komunikaci mezi klientem a serverem s sebou nese celou řadu problémů. Musíme vyřešit jakým způsobem budeme data přenášet, jak data budeme serializovat/deserializovat a jak budeme data validovat. Všechny tyto problémy řeší knihovny z kategorie middleware. Použití takové knihovny nám navíc zajistí typovou bezpečnost a zlepší vývojářskou zkušenost. Abychom neztratili interoperabilitu serveru s ostatními aplikacemi, použijeme knihovnu, která umožní vytvořit

i REST API. Mezi populární volby řešení patří knihovny gRPC od firmy Google, json-rpc-2.0 implementující standard JSON-RPC 2.0 a tRPC. Knihovna gRPC však nefunguje v prohlížeči [17], takže ji pro tento projekt nemůžeme použít. Knihovna tRPC má skvělou podporu pro TypeScript a je kompatibilní s knihovnou NextAuth.js. Knihovna tRPC je oproti knihovně json-rpc-2.0 značně populárnější, nabízí více funkcí a má aktivnější vývoj. Proto jsem se nakonec rozhodl použít knihovnu tRPC.

3.5.4 Backend aplikace

nginx Veškeré požadavky na server budou směrovány přes reverse proxy. To nám umožní konfigurovat routing, SSL certifikáty apod. Mezi zvážené možnosti patří Apache HTTP Server a nginx. Nginx má přehlednější dokumentaci a čitelnější formát konfigurace. Nginx má navíc největší podíl z počítačů na internetu [16].

NextJS Důvody popsány v sekci 3.5.2.

Form.io Důvody popsány v sekci 3.5.1.

MongoDB Form.io server podporuje pouze MongoDB pro ukládání dat.

PostgreSQL Potřebujeme databázi pro ukládání dat o úkolech a také nedokončené odpovědi na dotazníky. Chceme řešení, které je dostatečně stabilní a jeho licence umožňuje komerční použití. Výběr konkrétního řešení není příliš důležitý z následujících důvodů. Naše nároky na databázový systém jsou poměrně nízké a navíc budeme používat abstrakci nad databází, která nám umožní databázový systém kdykoliv vyměnit. PostgreSQL patří mezi nejlepší řešení splňující všechny naše požadavky.

Prisma Object relational mapper (ORM) je software používaný pro překlad mezi objektovou reprezentací dat a reprezentací, kterou používají databázové systémy [10]. Mapování mezi objekty a tabulkami relací je problém, který je těžké vyřešit obecně, a proto ORM knihovny často generují neoptimální SQL dotazy. Tyto dotazy mohou více či méně celou aplikaci zpomalovat. Tyto knihovny však umožňují zrychlit vývoj aplikace a zjednodušit práci s databází. Díky tomuto nástroji se můžeme vyhnout psaní velkého množství *boilerplate* kódu a soustředit se na implementaci business logiky. Mezi populární volby patří knihovny Drizzle ORM, TypeORM a Prisma. Knihovna Prisma je nejvyspělejší a nejpopulárnější z těchto knihoven co se týče počtu stažení za týden z npm registry. Knihovna je navíc vyvíjen komerční firmou, což zvyšuje její dlouhodobou udržitelnost.

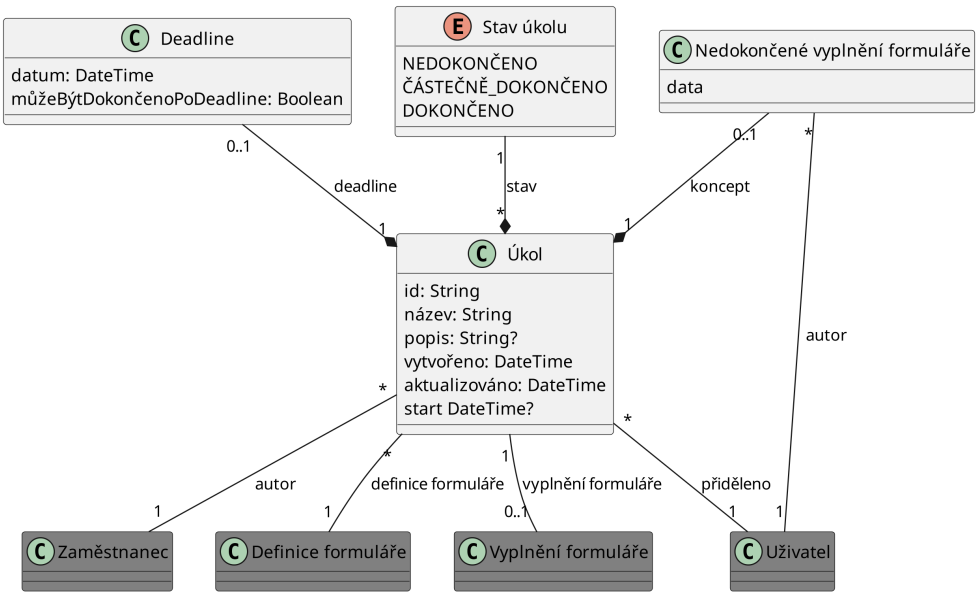
3.6 Ukládání dat

Tato sekce se věnuje způsobu ukládání dat v aplikaci. V podsekci 3.6.1 popíšeme logický datový model. Následně navrhneme řešení správy dat o nedokončených vyplnění formuláři v podsekci 3.6.2.

3.6.1 Logický datový model

Nyní popíšeme jak data budete ukládat na logické úrovni. Logický datový model je zobrazen na obr. 3.5 a používá notaci UML. Nebudeme se zabývat detaily ukládání dat o uživatelích a formulářích, jelikož o to se stará software Form.io. Entity, jejichž reprezentací se nechceme zabývat, jsou označeny na obrázku tmavě šedou barvou. Budeme se zabývat ukládáním úkolů a částečných vyplnění formulářů. Zvolíme relační datový model pro modelování těchto dat. Třídy v diagramu budou reprezentovat tabulky v databázi.

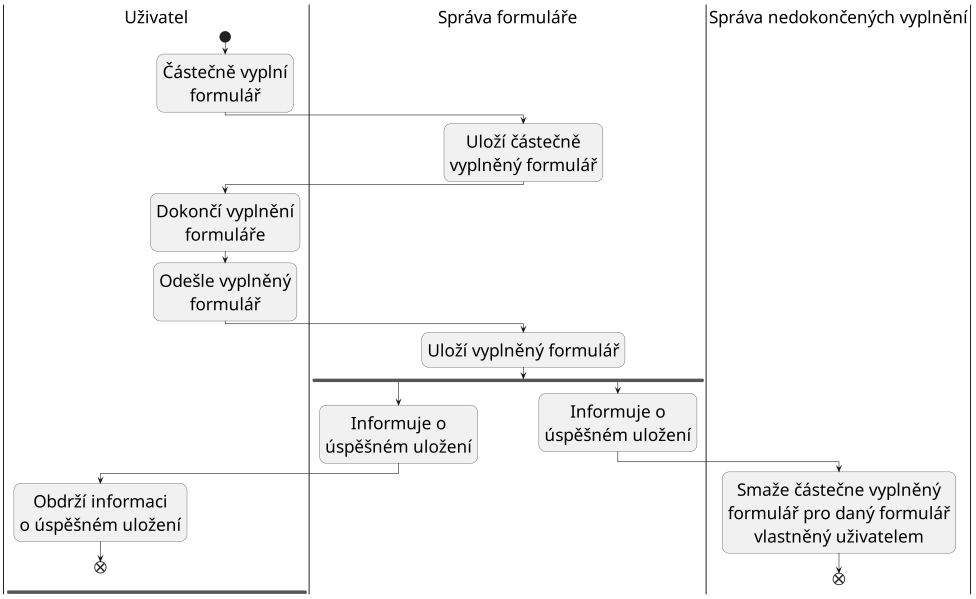
Pro stav úkolu budeme potřebovat místo tradičních dvou stavů (dokončený a nedokončený) stavy tři. Důvody k tomuto rozhodnutí jsou blíže popsány v sekci 4.5.



Obrázek 3.5: Logický datový model

3.6.2 Správa dat o nedokončených vyplnění formulářů

Plnitel si může vytvořit koncept pro libovolný dotazník. Při odevzdání dotazníku informuje systém spravující formuláře automaticky pomocí webhook systému systém spravující nedokončené vyplnění formulářů. Tento proces je zobrazen na obr. 3.6 jako diagram aktivity v notaci UML. Díky tomu nezůstávají nedokončené vyplnění formulářů v systému navždy. Pokud však tento webhook selže, nedokončené vyplnění formuláře v systému zůstane. Tato situace je velmi nepravděpodobná, takže ji nebudeme řešit. Pokud by se v průběhu užívání ukázalo, že se jedná o větší problém než se předpokládalo, lze zavést omezení na životnost konceptu. Například bychom mohli ukládat pouze koncepty, které byly použity v posledních 30 dnech. Tím bychom zajistili, že i pokud webhook selže, tak se koncept v systému nezachová déle než 30 dní.



Obrázek 3.6: Správa nedokončených vyplnění dotazníků

4. Vývojová dokumentace

Na začátku kapitoly popíšeme zvolené nástroje pro vývoj aplikace (sekce 4.1). Také jsou zde popsány i řešení menších problémů, které si vyžadují vysvětlení jako je řešení autentizace, soukromí uživatelů apod.

4.1 Nástroje pro vývoj

V této sekci popíšeme nástroje a knihovny a vysvětlíme proč jsme se rozhodli je použít. Projekt využívá continuous integration (CI), což je postup, kdy vývojáři projektu často (alespoň denně) integrují své změny do kódu projektu. Tyto změny jsou provázeny kontrolní kompilací projektu a testováním, které pomáhají předejít chybám [24]. Projekt využívá CI konkrétně pro automatizování formátování kódu, statickou analýzu kódu, generování dokumentace a testování. Jednotlivé části CI pipeline jsou detailně popsány níže.

GitHub Workflows Na základě dobré předchozí zkušenosti byl pro realizaci continuous integration zvolen nástroj Github Workflows.

GitHub Issues Pro issue tracking byl zvolen nástroj GitHub Issues, který je přímo integrován do platformy GitHub.

Prettier Standardizace formátování kódu zlepší čitelnost kódu a zrychlí vývoj [23]. Manuální formátování kódu je však zdlouhavé a proto použijeme nástroj, který tento proces automatizuje. Na základě osobní zkušenosti byl zvolen nástroj Prettier s jeho výchozím nastavením.

ESLint Statická analýza kódu pomáhá předejít mnoha chybám a zvyšuje kvalitu kódu. Pro tyto účely jsem zvážil nástroj ESLint a jshint. Na základě pozitivní osobní zkušenosti s nástrojem ESLint a jeho vyššímu počtu týdenních stažení na npm registry byl zvolen nástroj ESLint [2, 7].

CodeQL Statická analýza kódu může pomáhat předejít i bezpečnostním chybám. Proto jsme v projektu nasadili nástroj CodeQL, který je schopen detekovat bezpečnostní chyby pomocí sémantické analýzy kódu. Každá změna v kódu musí projít kontrolou tímto nástrojem než je sloučena do hlavní větve, pro zajištění maximální bezpečnosti.

Dependabot Pravidelně probíhá kontrola všech závislostí nástrojem Dependabot, který upozorňuje na známé bezpečnostní chyby v závislostech.

TypeDoc Tento nástroj implementuje generování dokumentace z dokumentačních komentářů dle standardu TSDoc. Alternativou je DocFX, který je napsaný v jazyce C#. Tento nástroj nemá npm balíček, což komplikuje integraci do našeho projektu. Pro tento projekt byl zvolen TypeDoc, jelikož implementuje stejný standard a jeho použití je v našem případě jednodušší.

Vitest Vývoj větších aplikací si vyžaduje testování kritických částí aplikace, abychom předešli špatné uživatelské zkušenosti a bezpečnostním chybám. Původně byla pro testování použita knihovna Jest, ale konfigurace pro náš

projekt byla neúspěšná. Složité chybové hlášky a nedostatečná dokumentace vedly na vykoušení jiné knihovnu. Byla zvolena knihovna Vitest, kterou bylo jednoduché nakonfigurovat zejména díky kvalitní dokumentaci.

Docusaurus Součástí požadavků na tento projekt je také dokumentace. Dokumentace usnadňuje vývojářům orientaci v kódu a zachycuje důležitá rozhodnutí. Na začátku projektu byla využívána Github Wiki, ale časem se ukázalo, že je pro tento projekt nevhodná. Github Wiki chybí podpora pro *diagrams-as-code*[1] a také nelze zahrnout automaticky generovanou dokumentaci z kódu. Proto byly zváženy další alternativy jako MkDocs, GitBook a Docusaurus. Docusaurus působí moderně, jednoduše a všechny jeho funkce jsou zdarma. Docusaurus má mnoho rozšíření jako například podporu pro diagramy jako kód. Docusaurus má oficiální plugin pro MermaidJS, což je nástroj pro tvorbu diagramů. Ukázalo se, že MermaidJS není vhodný pro náš projekt, protože nemá dobrou podporu diagramů C4 modelu, které používáme pro dokumentaci architektury. Díky velkému ekosystému nástroje Docusaurus však existuje i komunitní plugin pro podporu PlantUML diagramů, který má skvělou podporu diagramů C4 modelu. Integrace PlantUML diagramů s GitBook je naopak poměrně komplikovaná, proto GitBook nebyl použit. MkDocs a Docusaurus jsou srovnatelné, ale Docusaurus má některé funkce navíc jako např. použití React komponent pro tvorbu interaktivních částí. Přestože tyto funkce pravděpodobně nevyužijeme, nakonec byl zvolen nástroj Docusaurus.

4.2 Soukromí uživatelů

Aplikace pracuje s citlivými daty a proto je pro nás bezpečnost na prvním místě. Tato sekce se věnuje krokům, které jsme podnikli pro zajištění maximální bezpečnosti a soukromí uživatelů.

Uživatelé v systému vystupují pod náhodně generovaným ID, aby i v případě úniku dat nebylo možné získat citlivé informace o uživatelích.

Po všech uživatelích vyžadujeme silná hesla, která jsou bezpečně uložena. Pro řešení autentizace byly zváženy i bezheslové metody autentizace jako použití biometrických údajů, magic links či jednorázových tokenů [22]. Tyto metody však nelze v naší aplikaci využít z následujících důvodů. Chybí nám informace jako e-mail, telefonní číslo apod. Nechceme po uživateli vyžadovat vlastnictví specializovaného hardware jako bezpečnostní token či skener otisku prstu.

Webové rozhraní používající knihovnu NextJS ve výchozím nastavení sbírá anonymní data o uživatelích [8]. Na dotaz zadavatele byl sběr těchto dat vypnutý pro zvýšení důvěry uživatelů v platformu (viz produkční Dockerfile kontejneru `web-app`).

Dle vyjádření správců projektu Form.io software Form.io nesbírá žádná data o uživatelích [3].

[1] Diagrams-as-code je metoda pro tvorbu diagramů použitím textových definic zapsaných v doménově specifické jazyka [21]

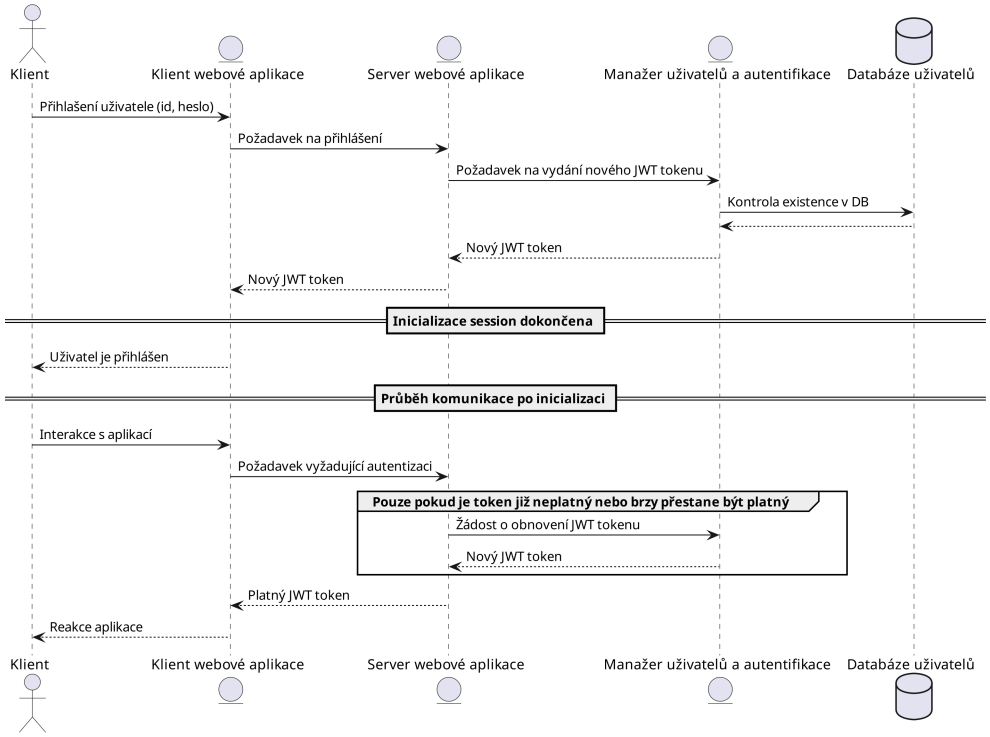
4.2.1 GDPR

Při autentizaci uživatele používáme cookies pro správu session. Ukládání dat do úložiště klienta spadá pod GDPR, proto je nutno ověřit, jaké jsou naše povinnosti. Dle doporučení Evropské Unie (sekce 3.2) pro autentizační cookies je potřeba pro *persistentní* cookies upozornit uživatele o jejich užití. Knihovna NextAuth.js, kterou používáme má podporu pouze pro persistentní cookies (viz issue), tedy musíme uživatele řádně upozornit. Pro tento účel jsem zvolil knihovnu react-cookie-consent.

GDPR také vyžaduje, aby uživatelé měli možnost smazat svá data. Existuje však výjimka ohledně ukládání vědeckých dat. Většina dat v systému bude vědeckého charakteru, ale pro část dat toto neplatí. Smazaní dat, která nebudou určena pro výzkum, bude možné po vyžádání uživatelem u provozovatele systému a bude se řešit manuálně.

4.3 Autentizace

Jak bylo vysvětleno v sekci 4.2, tak uživatelé se přihlašují pomocí náhodně generovaného ID a hesla. Administrátoři se přihlašují pomocí e-mailu a hesla. Pro autentizaci se používá knihovna NextAuth.js, která vykonává kód jak na klientovi, tak na serveru webové aplikace. Tato knihovna má vlastní session management a vydává JWT tokeny. Aby klient mohl pracovat se systémem spravující formuláře, je potřeba získat JWT token systému spravující formuláře. Token systému spravující formuláře je vložen do tokenu vydávaného knihovnou NextAuth.js. Tento systém zařizuje, že webová aplikace nemusí při každém požadavku od klienta komunikovat s systémem spravující formuláře, ale má vlastní session. Kontrola práv uživatele při přístupu na ochráněnou stránku se kontrola provádí v middleware NextJS serveru.



Obrázek 4.1: Komunikace při autentizaci uživatele

4.4 Konfigurace a modifikace Form.io

Submodul `/src/formio/` obsahuje fork projektu Form.io, kde bylo provedeno několik úprav. Úpravy jsou poměrně malé a jejich cílem je přizpůsobit software našim potřebám. Všechny změny jsou popsány v této sekci. Dále jsou zde popsány některé části konfigurace Form.io.

4.4.1 Modifikace

Inicializace projektu

První úprava se týká inicializace instance při prvním startu. Nepodařilo se najít způsob, jak nastavit inicializaci projektu tak, aby se automaticky vytvořili všechny potřebné zdroje (klient, pacient, zaměstnanec, apod.), role (admin, zaměstnanec, apod.). Proto jsem upravil kód inicializace tak, aby se načetla z souboru `project.json` z kořenu repozitáře obsahující fork. Původně se načítala z `project.json` v build složce, která však není uložena ve verzovacím systému. Tato úprava byla nutná, jelikož nelze vytvářet nové role po inicializaci projektu a konfigurační soubor je nutné zahrnout do verzovacího systému.

Webhook action

Webhook je koncept, který umožňuje poslat HTTP požadavek na zadanou URL jako reakci na určitou událost. V našem případě se jedná o události jako je vytvoření nebo smazání formuláře. V open-source verzi Form.io software je tato funkce podporována, ale neumí přeposílat hlavičky HTTP dotazu, který

požadavek inicioval. Tuto funkcionalitu potřebujeme, jelikož chceme přeposílat autentizační hlavičky při komunikaci mezi systémem spravující formuláře a systémem spravující úkoly uživatelů. Detaily ohledně této komunikace jsou popsány v sekci 4.5. Proto jsem upravil tuto funkci tak, aby bylo možné přeposílat autentizační hlavičky.

Imutabilita ID uživatele

Form.io software lze rozdělit na dvě části. První část je systém pro správu formulářu a sbírání odpovědí na tyto formuláře. Druhá část je systém pro správu uživatelů. Systém správy formulářů je použit i pro správu uživatelů systému. V systému jsou vytvořeny registrační formuláře, které umožňují tvorbu uživatelských účtů. Data o uživatelských účtech jsou pak uložena v uložišti odpovědí na tyto registrační formuláře. V systému existují samosatné registrační formuláře pro různé role uživatelů. My potřebujeme zajistit imutabilitu identifikátoru každého uživatele, který je uložen v položce `data.id`. Motivaci k této změně naleznete v sekci 4.3. Databáze MongoDB, která je použita pro ukládání dat o uživatelích neumí zajistit imutabilitu ukládaných dat. Form.io však vždy přistupuje k databázi uživatelů skrze knihovnu Mongoose. Tato knihovna umí zajistit imutability zvolených položek. Mongoose nastavíme tak, aby zajistil imutabilitu položky `data.id` (ID uživatele). Pokud se uživatel pokusí změnit své ID v rámci úpravy svého profilu, tak se změna neprovede. Nevýhodou tohoto řešení je, že vytváří omezení na položky `data.id` ve všech objektech reprezentující odevzdání, tedy i v těch, které nejsou uživatelské účty.

Alternativní řešení by bylo vytvořit autorizační proxy mezi klientem a Form.io serverem či mezi Form.io serverem a databází uživatelů. Toto řešení by pravděpodobně vyžadovalo další kontejner a má netriviální implementaci. Implementace by navíc měla nízkou odolnost vůči změnám. Pokud bychom měli více koncových bodů API, které umožňují modifikovat entitu uživatele, bylo by potřeba myslet na zabezpečení všech koncových bodů. V případě přidání nového bodu, by bylo vždy potřeba myslet na úpravy autorizační proxy. Z těchto důvodů byla tato alternativa zavrhnuta a bylo použito řešení popsáno v předchozím odstavci.

4.4.2 Konfigurace

Tato podsekce popisuje, jak byl software Form.io nakonfigurován pro náš projekt.

Konfigurace zdrojů (resources)

Form.io pracuje s konceptem zdrojů, které lze zjednodušeně chápat jako kolekce objektů v databázi. Detailní popis tohoto konceptu lze nalézt v dokumentaci Form.io. Pro nás je nyní důležité, že každý objekt patří do právě jednoho zdroje. Toto omezení platí, protože používáme bezplatnou verzi Form.io. Uživatelé jsou reprezentováni jako objekty patřící do nějakého zdroje. Každý zdroj je svázán se sadou rolí. Role uživatele jsou tedy určeny zdrojem, kterému uživatel patří. Proto je potřeba vytvořit zdroj pro každou kombinaci rolí, které může uživatel mít. Vzhledem k tomu, že jsou tyto zdroje oddělené, nelze při tvorbě uživatelského účtu jednoduše validovat, že má každý uživatel unikátní ID pro přihlášení.

Abychom předešli zmatení uživatelů, v uživatelském rozhraní vždy k ID přidáme i roli, která je uživateli přiřazena.

Alternativně bychom mohli každé existující kombinaci rolí přidat prefix do ID uživatele. Toto řešení však značně zhoršuje uživatelskou přívětivost a proto nebylo použito.

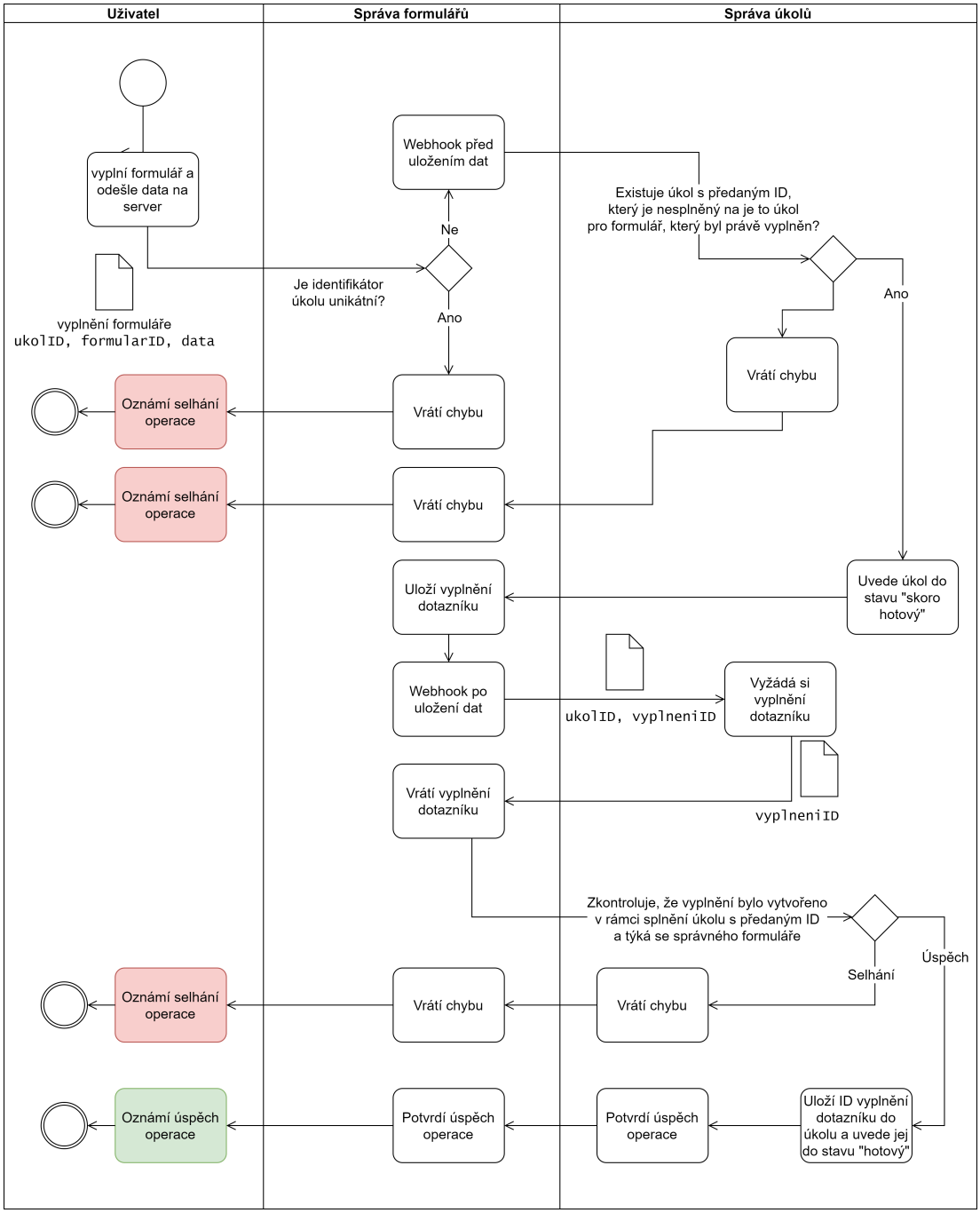
4.5 Propojení správy úkolů a správy formulářů

Náš systém obsahuje komponentu spravující úkoly a komponentu spravující formuláře. Každý úkol je spojen s formulářem, který má uživatel vyplnit. Potřebujeme v našem distribuovaném systému zajistit, aby uživatel mohl vyplnit dotazník pouze tehdy, když má úkol na jeho vyplnění. Popíšeme dvě možnosti řešení tohoto problému a následně zvolíme jednu z nich.

První možností řešení je, že uživatel vyplní dotazník a *odešle se požadavek na komponentu spravující formuláře*, kde je připraven webhook, který proběhne *před* uložením výsledku, kde proběhne kontrola existence úkolu na vyplnění tohoto dotazníku. Pokud úkol existuje, tak se vyplnění dotazníku uloží společně s identifikátorem úkolu a také se splní úkol a uloží se k němu identifikátor vyplnění dotazníku. V případě, že úkol neexistuje, tak se vyplnění dotazníku neuloží a uživatel je informován o chybě. Dotazník obsahuje skryté pole, které obsahuje identifikátor úkolu v rámci, kterého byl dotazník vyplněn. Tím je zajištěno, že vyplnění dotazníku drží informaci o identifikátoru úkolu v rámci kterého byl vyplněn.

Zvažme nyní druhou možnost řešení. Uživatel vyplní dotazník a *odešle se požadavek na komponentu spravující úkoly*, která za něj dotazník odevzdá pokud má uživatel úkol na jeho vyplnění. Situace se při tomto řešení komplikuje, jelikož bychom museli zajistit, že všechny požadavky na vyplnění dotazníku musí jít skrze komponentu spravující úkoly. Museli bychom tedy zakázat, aby uživatel odevzdal dotazník přímo do systému spravující formuláře. Kdybychom to neudělali, tak nejsme schopni zajistit integritu sesbíraných dat. Toto omezení bychom mohli zajistit úpravou konfigurace reverse proxy, kterou prochází všechny požadavky z vnějšího světa, nebo úpravou systému spravující formuláře. První zmíněný způsob by však nezajistil ochranu před požadavky z vnitřní sítě. Pro ochranu před požadavky z vnitřní sítě je lepší druhý způsob. Systém spravující formuláře je řešením třetí strany a vyžadovalo by to zásah do jejich kódu.

Zvolíme první alternativu, jelikož je jednodušší na implementaci a má dobré vlastnosti. Nyní si rozmysleme detaily implementace. Chceme zajistit, aby se nám nepodařilo vytvořit více vyplnění dotazníku pro stejný úkol. Proto formulář, který bude zadáván jako úkol pro plnitele, nastavíme tak, aby zajistil unikátnost hodnoty skryté položky obsahující identifikátor úkolu. Nyní popíšeme celý proces vyplnění dotazníku formou diagramu (Obr. 4.2) používající Business Process Model and Notation.

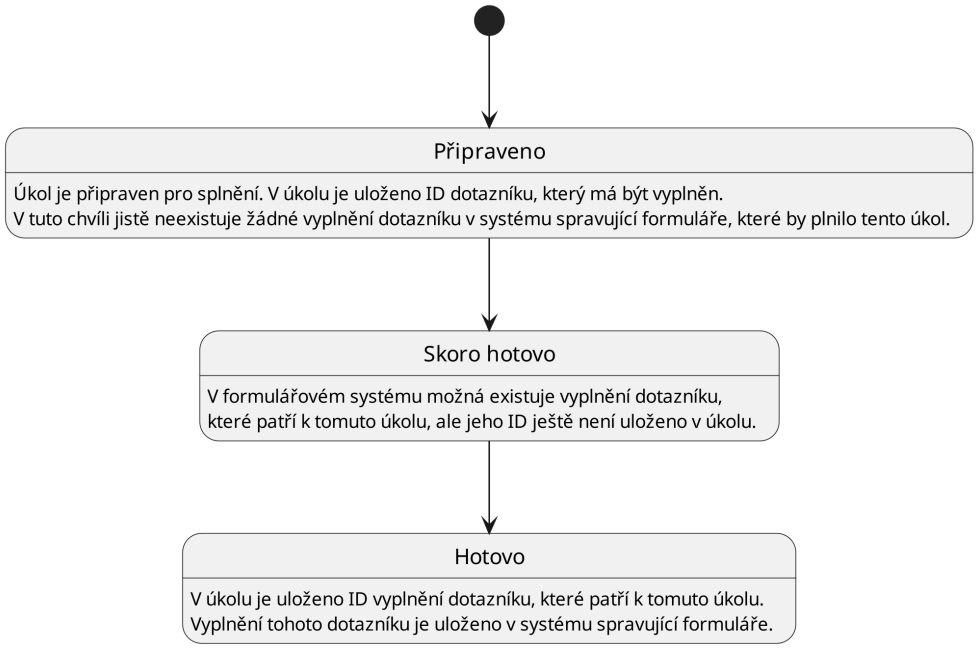


Obrázek 4.2: Diagram integrace systémů spravující úkoly a systému spravující formuláře

Požadavek na uložení vyplnění dotazníku spustí webhook. V rámci reakce na tento webhook uvedeme úkol do stavu `skoro hotový`. Tento stav slouží k tomu, aby nebylo možné vytvořit další vyplnění dotazníku v rámci stejného úkolu. Tento mechanismus se nazývá *semantic lock counter-measure* [15]. Stavy úkolu a přechody mezi nimi jsou popsány stavovým diagramem v jazyce UML na obr. 4.3.

Nyní popíšeme, jak je tento mechanismus využíván v našem systému. Pokud požadavek na uložení vyplnění dotazníku spustí webhook, který proběhne před uložením do systému spravující formuláře, a následně selže operace uložení vypl-

není dotazníku v systému spravující formuláře, tak úkol navždy zůstane ve stavu `skoro hotový`. Pokud bychom se z tohoto stavu chtěli dostat, potřebovali bychom implementaci ság, která by zajistila provedení kompenzačních transakcí. Popsaná situace je však natolik výjimečná, že se v tuto chvíli problematice věnovat nebudeme. Vystačíme si s tím, že systém spravující formuláře nikdy nebude obsahovat vyplnění dotazníku, které vzniklo bez zadání úkolu.



Obrázek 4.3: Stavový diagram úkolů

Po uložení vyplnění dotazníku se zavolá druhý webhook, který zařídí uložení informace o identifikátoru vyplnění do úkolu. Vzhledem k tomu, že se jedná o veřejně dostupný endpoint, tak je nutné ověřit poskytnutá data. Musíme si tedy z úkolového systému vyžádat vyplnění dotazníku a zkontrolovat všechny náležitosti. Pokud by se někdo pokusil spárovat úkol s formulářem, který je ve stavu `skoro hotový` na jiné vyplnění dotazníku než to, které způsobilo přechod úkolu do stavu `skoro hotový`, tak operace jistě selže. Kdyby se někdo pokusil spárovat úkol s vyplněním dotazníku, který neodpovídá dotazníku zadaný úkolem, tak operace jistě selže. V případě, že by se někdo pokusil spárovat jiné vyplnění stejného dotazníku, tak se mu to také nepovede. Ono jiné vyplnění dotazníku by muselo mít jiný identifikátor úkolu v skrytém poli, protože systém spravující formuláře zajišťuje unikátnost těchto hodnot. Jelikož víme, že již existuje vyplnění dotazníku, které má identifikátor úkolu v skrytém poli stejné jako identifikátor úkolu, který se právě snažíme napárovat, tak víme, že jakékoliv jiné vyplnění stejného dotazníku bude mít jiný identifikátor úkolu v skrytém poli.

Autentizace webhooků HTTP požadavky, které dělá systém spravující formuláře při tvorbě odevzdání formuláře na systém spravující úkoly, musí obsahovat autentizační hlavičky. První možností je poskytnout systému spravující formuláře speciální přístupový token. Toto však nelze provést pouze pro některé webhooky bez většího zásahu do zdrojového kódu systému spravující formuláře, ale pouze pro všechny webhooky najednou. Navíc to situaci zbytečně komplikuje.

Druhá možnost je využití JWT tokenu z požadavku na vytvoření odevzdání formuláře, který je přijat systémem spravující formuláře. Toto je dobrá možnost, ale open-source verze systému spravující formuláře tuto funkci nemá. Nicméně není těžké ji do něj přidat.

Třetí možností je vyhnout se autentizaci webhooku pomocí přihlášení. Místo toho bychom mohli vytvořit endpoint komponenty spravující úkoly, který je přístupný pouze z komponenty spravující formuláře. Toto však znamená netriviální úpravy síťové infrastruktury. Tyto úpravy jsou poměrně špatně udržovatelné.

Použijeme druhou možnost, jelikož má jednoduchou implementaci. Provedené úpravy systému spravující formuláře jsou popsány v sekci 4.4.

5. Testování aplikace

Serverová část aplikace je testována pomocí automatizovaných unit testů. Testujeme pouze veřejné rozhraní všech modulů. Objekt fungující jako proxy databáze byl nahrazen mock objekty. V testech kontrolujeme, zda-li se volají konkrétní metody na mock objektech s očekávanými parametry. Ačkoliv je vyše popsaný způsob doporučeným přístupem dle dokumentace object-relational mapping knihovny Prisma, v našem případě se neosvědčil.

Kontrola volání konkrétních metod vytváří obrovskou závislost na vnitřní implementaci testovaných metod. Testy jsou velmi těžko udržovatelné a navíc poměrně dlouhé a složité. Kdybych začínal znovu, tak bych zvolil jinou dekompozici nebo bych zvážil použití in-memory databáze[1].

V tabulce 5.1 je zobrazeno pokrytí serverové části automatizovanými unit testy.

Výrazy	**Větve**	**Funkce**	**Řádky**
90.85 %	72.91 %	100 %	90.85 %

Tabulka 5.1: Pokrytí serverové části testy

[1] In-memory databáze je databáze spoléhající primárně na vnitřní paměť pro ukládání dat [6]

6. Administrátorská příručka

Tato kapitola popisuje, jak spravovat aplikaci z pohledu administrátora. První jsou popsány kroky pro instalaci a konfiguraci aplikace v sekci 6.1. Následně je popsáno spuštění aplikace v sekci 6.2. V sekci 6.3 jsou popsány kroky pro tvorbu uživatelských účtů. Nakonec jsou popsány přístupy k správcovským rozhraním jednotlivých částí aplikace v sekci 6.4.

Obsah této kapitoly je k dispozici i v repozitáři projektu v souboru `README.md`.

6.1 Instalace a konfigurace

Pro instalaci aplikace je potřeba mít nainstalovaný Docker a Git.

Celý projekt je zastřešen git repozitářem mental-health-monitoring-platform. Tento repozitář používá git submoduly, jelikož zdrojový kód platformy Form.io je v jiném repozitáři. Jedná se o repozitář mental-health-monitoring-platform-formio, což je fork repozitáře formio, který obsahuje oficiální zdrojový kód platformy Form.io. Pro instalaci aplikace je potřeba naklonovat repozitář mental-health-monitoring-platform včetně submodulů, což lze provést následujícím příkazem:

```
git clone --recurse-submodules \
  https://github.com/PatrikTrefil/mental-health-monitoring-platform.git
```

Klonování a inicializaci submodulů lze provést i pomocí následující sekvence příkazů:

```
git clone \
  https://github.com/PatrikTrefil/mental-health-monitoring-platform.git
cd mental-health-monitoring-platform
git submodule init # initialize configuration file
git submodule update # fetch submodule data
```

Před spuštěním aplikace je potřeba dodat `.env` soubor do kořenu repozitáře pro konfiguraci prostředí. Syntax konfiguračního souboru je popsána v dokumentaci Docker compose. Ukázkovou konfiguraci najdete v `.env.example`. Zde je seznam proměnných prostředí, které je potřeba nastavit, a jejich význam:

MONGO_INITDB_ROOT_USERNAME
: Proměnná obsahuje uživatelské jméno pro přihlášení do MongoDB jako superuživatel (root).

MONGO_INITDB_ROOT_PASSWORD
: Proměnná obsahuje heslo pro do MongoDB jako superuživatel (root).

FORMIO_NODE_CONFIG
: Proměnná obsahuje konfigurace Form.io aplikace. Hodnota by měla být ve formátu JSON. Výchozí hodnoty konfigurace jsou dostupné zde. Není potřeba nastavit všechny atributy konfiguračního objektu, ale pouze ty, které chceme přepsat.

FORMIO_ROOT_EMAIL
Proměnná obsahuje e-mail pro přihlášení do Form.io aplikace jako superuživatel (root).

FORMIO_ROOT_PASSWORD
Proměnná obsahuje heslo pro přihlášení do Form.io aplikace jako superuživatel (root).

FORMIO_MONGO_USER
Proměnná obsahuje uživatelské jméno pro přístup Form.io apliakace do MongoDB.

FORMIO_MONGO_PASSWORD
Proměnná obsahuje heslo pro přihlášení uživatele `FORMIO_MONGO_USER` do MongoDB.

DOMAIN_NAME
Proměnná obsahuje doménové jméno, na kterém bude aplikace dostupná (např. domena.cz, localhost).

NEXTAUTH_SECRET
Proměnná obsahuje klíč pro šifrování autentizačních tokenů.

Také je potřeba nainstalovat závislosti pro Form.io aplikaci:

```
cd src/formio && npm install
```

Nyní aplikaci můžeme spustit pomocí návodu v sekci 6.2. Poté je potřeba vytvořit první uživatelský účet, jak je popsáno v sekci 6.3.

6.2 Spuštění aplikace

Předpokládáme, že aplikace byla nainstalována a konfigurována podle návodu v sekci 6.1. Pro spuštění aplikace v produkčním módu spusťte následující příkaz z kořenového adresáře repozitáře:

```
docker compose up
```

Po spuštění příkazu bude aplikace dostupná na `http://localhost`.

Pokud se jedná o první spuštění aplikace, vytvořte prvního uživatele dle návodu v sekci 6.3.

Pro spuštění aplikace ve vývojovém módu použijte následující příkaz:

```
docker compose --file ./docker-compose.yml \
  --file ./docker-compose.dev.yml \
  up
```

Po spuštění příkazu bude aplikace dostupná na `http://localhost:8080`. Na jednotlivé služby aplikace se lze připojit i přímo. Mapování portů je definováno v `docker-compose.dev.yml`.

6.3 Tvorba uživatelských účtů

Tato sekce popisuje kroky pro vytvoření uživatelských účtů. Účty uživatelů libovolné role lze vytvořit pomocí webového rozhraní Form.io aplikace. Po prvním spuštění aplikace je doporučeno vytvořit uživatele s rolí správce dotazníků. Pro jednoduchost tvorby tohoto uživatele je připraven shell skript, jehož užití je popsáno v sekci 6.3.2.

6.3.1 Tvorba účtu pomocí webového rozhraní

Účet lze vytvořit pomocí webového rozhraní Form.io aplikace, která je dostupná na `/formio/` po spuštění aplikace. Po otevření rozhraní se zobrazí přihlašovací formulář. Do formuláře vyplníme přihlašovací údaje adminisrátorského účtu, které jsme definovali v konfiguračním souboru při instalaci aplikace (viz sekce 6.1). Po přihlášení je potřeba vyplnit formulář *Správce dotazníků registrace*. Uživatele *nevytvářejte* skrze modul *Resources*.

6.3.2 Tvorba účtu pomocí shell skriptu

Pokud máte k dispozici příkaz `curl`, tak uživatele *s rolí správce dotazníků* můžete vytvořit pomocí skriptu `./src/scripts/create_form_manager_user.sh`. Skript se spustí v interaktivním módu, pokud uživatel neposkytne všechny parametry programu. Parametry programu lze take předat pomocí argumentů programu. Parametry programu se odvíjí od konfigurace definované při instalaci aplikace (viz sekce 6.1). Dokumentaci nástroje lze získat pomocí příkazu:

```
./src/scripts/create_form_manager_user.sh --help
```

6.4 Správa aplikace

Aplikace se skládá z několika částí, které je potřeba spravovat. Zde je seznam částí aplikace a možné způsoby správy:

Reverse proxy Status stránka je dostupná na `/nginx_status`. Když chceme zjistit zda-li reverse proxy běží, můžeme udělat HTTP GET požadavek na `/health`. Pokud je návratový kód 200, tak reverse proxy běží.

Monitoring Monitoring služeb aplikace je dostupný na `/monitoring/`. Oficiální dokumentace k webovému rozhraní je dostupná online v anglickém jazyce zde. Monitorovací aplikace poskytuje i API, které je dostupné na `/monitoring/api`. Oficiální dokumentace k API je dostupná online v anglickém jazyce zde.

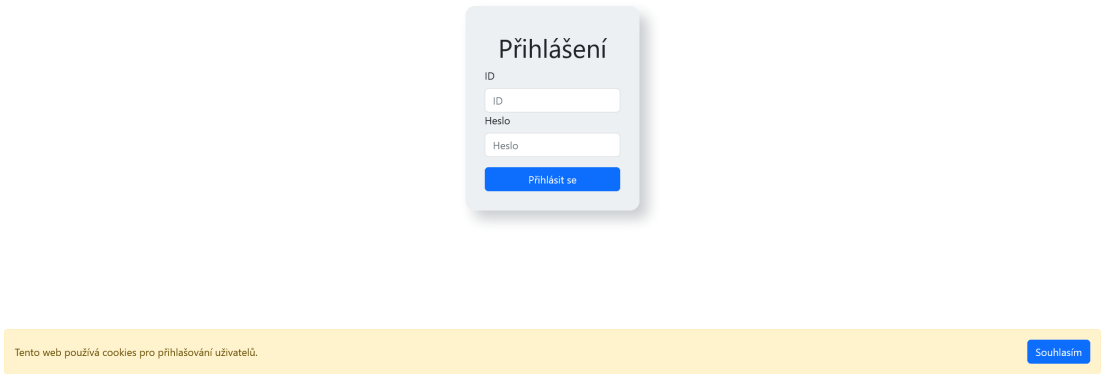
Form.io Webové rozhraní a API aplikace Form.io je dostupné na `/formio/` po spuštění aplikace. Oficiální dokumentace k webovému rozhraní je dostupná online v anglickém jazyce zde. Oficiální dokumentace k API je dostupná online v anglickém jazyce zde.

7. Uživatelská dokumentace

V této kapitole popíšeme užívání aplikace z pohledu zaměstnance a plnitele. Tyto role byly definovány v požadavku R-NR-1. První se podíváme na krok přihlášení, který je společný pro obě role, a následně na užívání aplikace. Zaměstnanci a plnitelé přistupují na odlišné části aplikace, které jsou přístupné pouze pro jejich roli. Z tohoto důvodu je popis uživání aplikace rozdělen do dvou sekcí dle role uživatele.

7.1 Přihlášení

Proces přihlášení začíná na přihlašovací obrazovce (Obr. 7.1), na kterou je uživatel přesměrován při vstupu do aplikace. Před samotným přihlášením je nutno schválit používání cookies (Detaily jsou popsány v sekci 4.2.1). Pro přihlášení uživatel zadá své přihlašovací údaje a stiskne tlačítko „Přihlásit se“. Následně je automaticky přesměrován na výchozí obrazovku pro svou roli. Pokud zaměstnanec nemá účet, musí si požádat o vytvoření účtu u jiného zaměstnance s již existujícím účtem a dostatečným oprávněním, či u administrátora aplikace. Pokud plnitel nemá účet, musí si požádat o vytvoření účtu u zaměstnance s již existujícím účtem a dostatečným oprávněním, či u administrátora aplikace. Tvorba účtu zaměstnancem je popsána v sekci 7.2.5.



Obrázek 7.1: Přihlašovací obrazovka

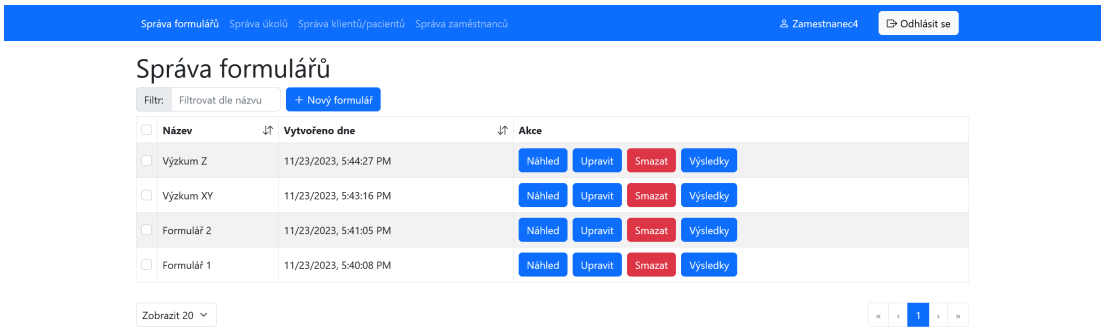
7.2 Užívání aplikace z pohledu zaměstnance

Tato sekce popisuje užívání aplikace z pohledu zaměstnance. Jsou zde popsány veškeré funkce, které byly identifikovány jako požadavky na funkcionalitu dostupnou zaměstnanci v kapitole 1. Předpokládáme, že zaměstnanec má již vytvořený účet a přihlásil se do aplikace, jak je popsáno v sekci 7.1, která se věnuje

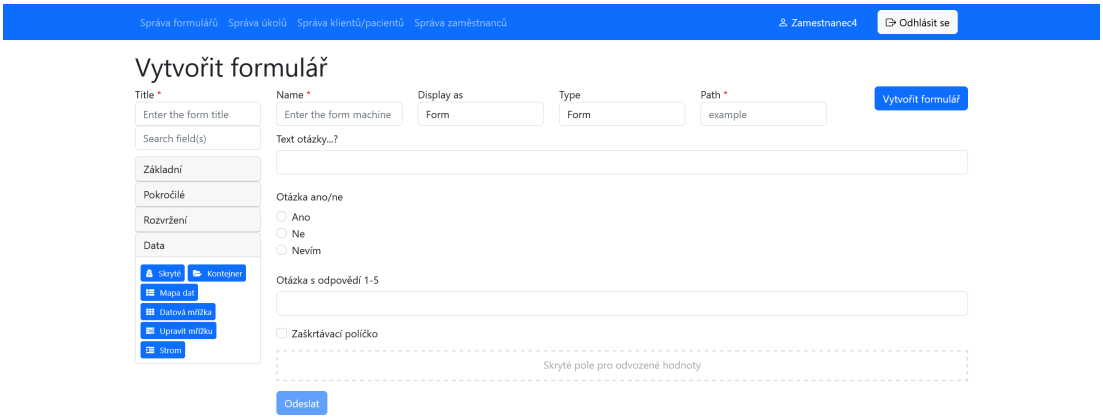
přihlášení a tvorbě účtu. Po přihlášení je zaměstnanec automaticky přesměrován na výchozí obrazovku pro zaměstnance (Obr. 7.2).

7.2.1 Správa formulářů

Pro zadání úkolu plniteli je nutno nejprve vytvořit formulář. Formuláře se vytváří v sekci „Správa formulářů“ (Obr. 7.2). Právo na tvorbu formulářů mají pouze zaměstnanci s rolí „Správce dotazníků“. Formuláře se vytváří pomocí tlačítka „Nový formulář“. Stisknutí tohoto tlačítka se dostaneme na stránku pro tvorbu formuláře (Obr. 7.3). Pro vytvoření je potřeba zadat název formuláře do pole Název a přidat jednotlivé otázky taháním prvků z levého panelu do prostoru pro tvorbu formuláře. Pole Identifikátor a Cesta se vyplní automaticky při zadání názvu a obvykle není třeba je měnit. Jak vypočítat odvozené hodnoty v rámci vyhodnocení formuláře je popsáno v podsekci 7.2.1. Detailní dokumentace k tvorbě formulářů je dostupná online v anglickém jazyce na tomto odkazu. Formulář uložíme do systému pomocí tlačítka „Vytvořit formulář“. Obrazovka pro správu formulářů (Obr. 7.2) nám nabízí několik dalších funkcí. Již existující formuláře můžeme upravovat pomocí tlačítka „Upravit“ nebo mazat pomocí tlačítka „Smazat“. Úpravy formulářů mohou ovlivnit již existující odevzdání formulářů a proto se doporučuje tuto funkci používat pouze pro drobné opravy.



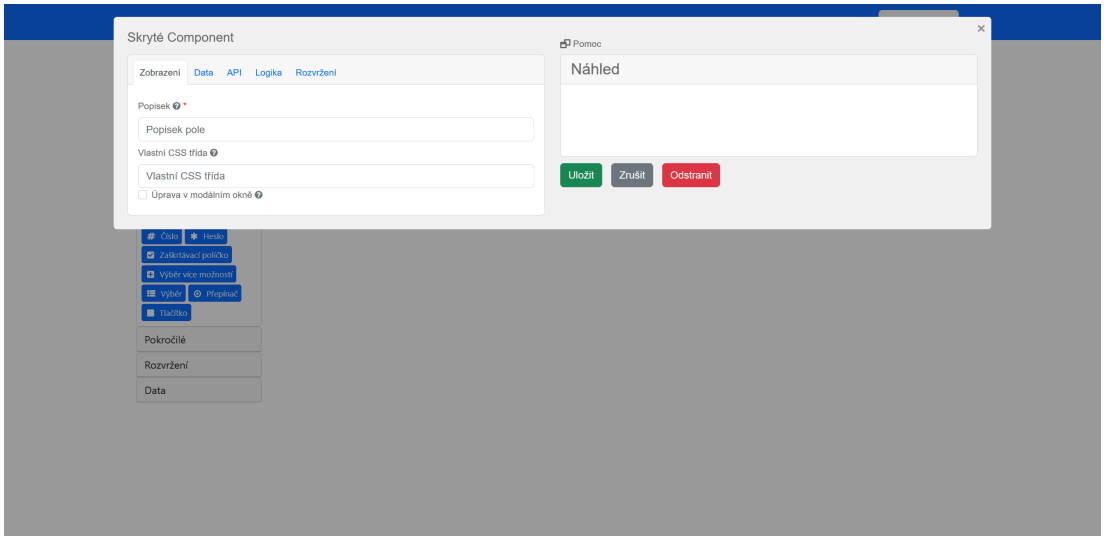
Obrázek 7.2: Správa formulářů aplikace



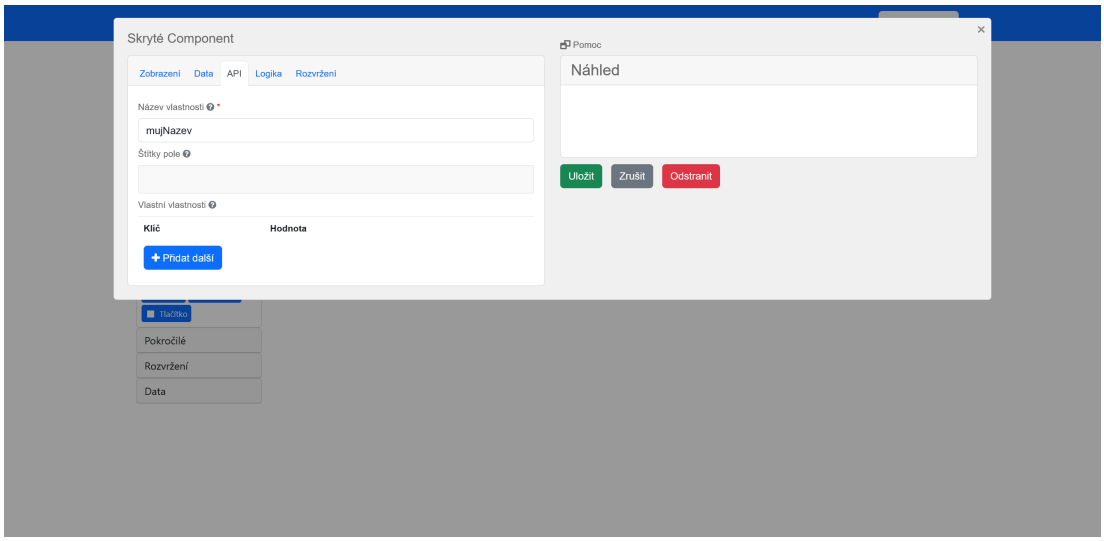
Obrázek 7.3: Tvorba formuláře

Výpočet odvozených hodnot

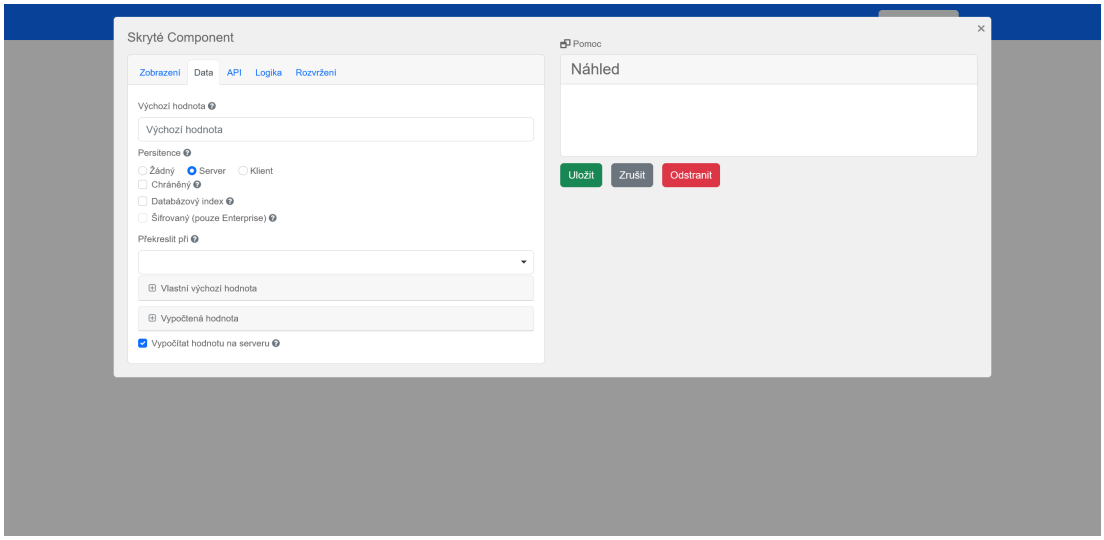
Pokud chceme formulář automaticky vyhodnotit na základě odpovědí plnitele, použijeme prvek „Skryté“ z kategorie „Data“ z levého panelu. Po přidání prvku se zobrazí jeho nastavení (Obr. 7.4). Vzorec pro výpočet hodnoty můžeme zadat do sekce „Vypočtená hodnota“ v kartě „Data“ (Obr. 7.6). Vzorec se zapisuje v programovacím jazyce JavaScript. Všechny hodnoty odpovědí jsou dostupné na objektu `data`. Vzorec používá názvy vlastností jako klíče na tomto objektu. Pro použití konkrétní můžeme použít tečkovou notaci `data.nazevVlastnosti` nebo notaci s hranatými závorkami `data["nazevVlastnosti"]`. Název vlastnosti lze pro každý prvek nastavit v menu nastavení v poli „Název vlastnosti“ v kartě „API“ (Obr. 7.5). Výsledek vzorce uložíme do proměnné `value`. Např. pro součet hodnoty odpovědí s názvy „a“ a „b“ bychom použili vzorec `value = data.a + data.b`. Pokud je nevhodné, aby uživatel viděl způsob výpočtu odvozené hodnoty nebo mohl získat vypočtenou hodnotu, tak je nutné v nastavení prvku zvolit možnost „Vypočítat hodnotu na serveru“ (Obr. 7.6). Nastavení prvku uložíme stisknutím tlačítka „Uložit“. Kdybychom chtěli nastavení prvku znovu upravit, tak se na obrazovku nastavení dostaneme stisknutím tlačítka s ozubeným kolečkem v pravém horním rohu prvku (Obr. 7.7).



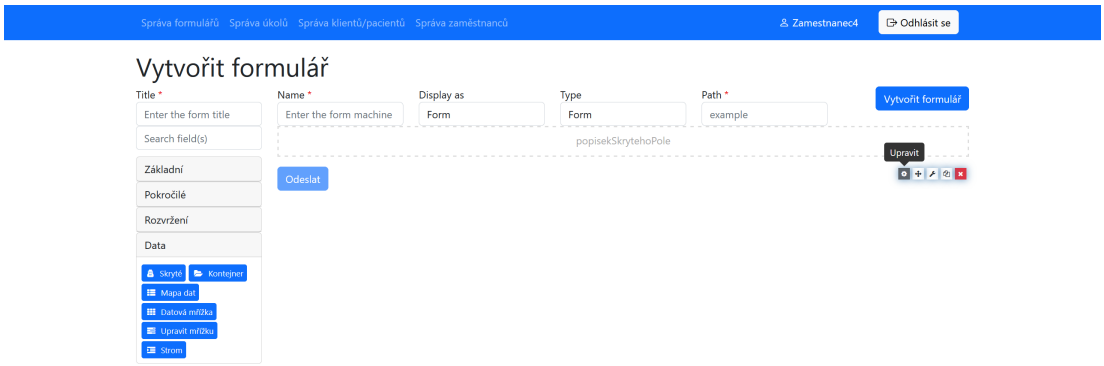
Obrázek 7.4: Karta Zobrazení v nastavení prvku Skryté



Obrázek 7.5: Karta API v nastavení prvku Skryté



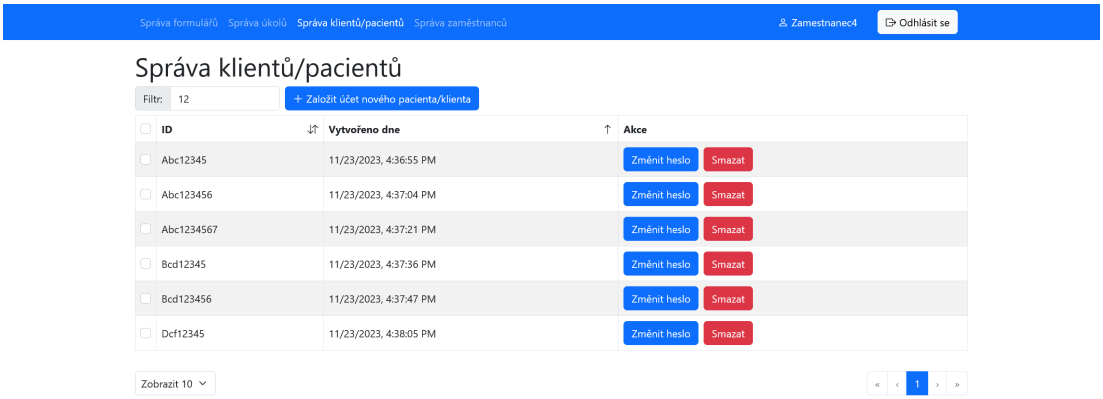
Obrázek 7.6: Karta Data v nastavení prvku Skryté



Obrázek 7.7: Zobrazení menu nastavení prvku

7.2.2 Správa plnitelů

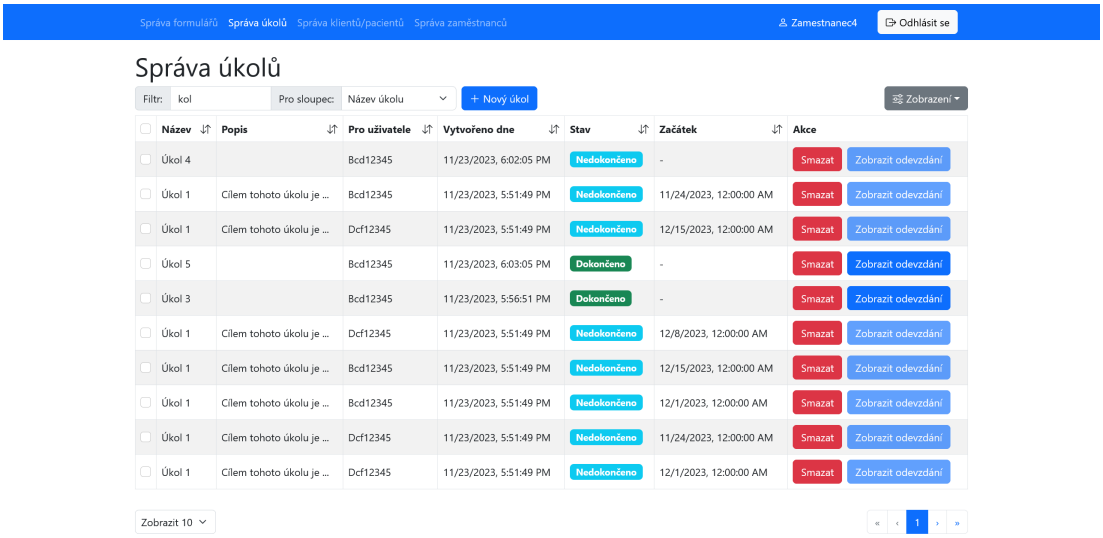
Pro zadání úkolu plniteli je nutno nejprve vytvořit uživatelský účet pro plnitele. Účty plnitelů se vytváří v sekci „Správa plnitelů“ (Obr. 7.8). Nový účet vytvoříme pomocí tlačítka „Založit účet nového klienta/pacienta“. Pro vytvoření účtu je potřeba zadat identifikátor plnitele, který je unikátní v rámci celé aplikace, a heslo. Plnitel si může heslo změnit po přihlášení do aplikace.



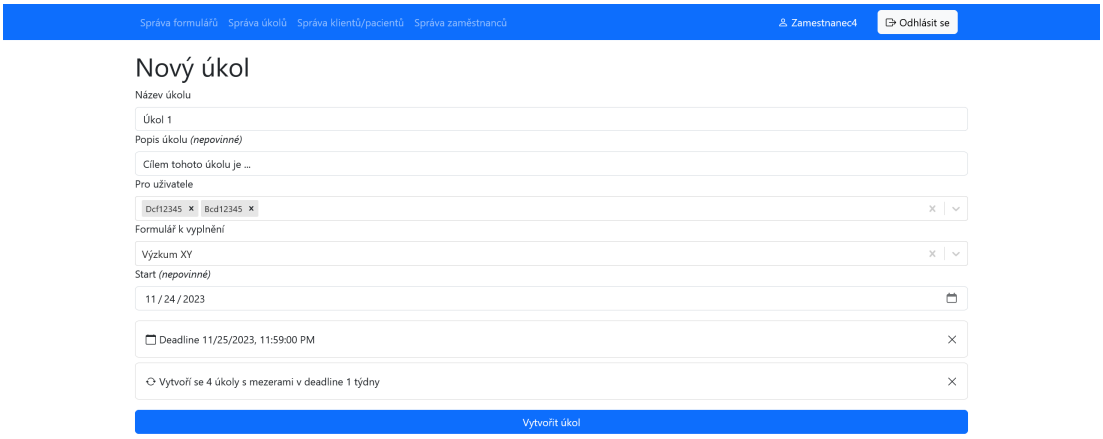
Obrázek 7.8: Správa účtů plnitelů

7.2.3 Správa úkolů

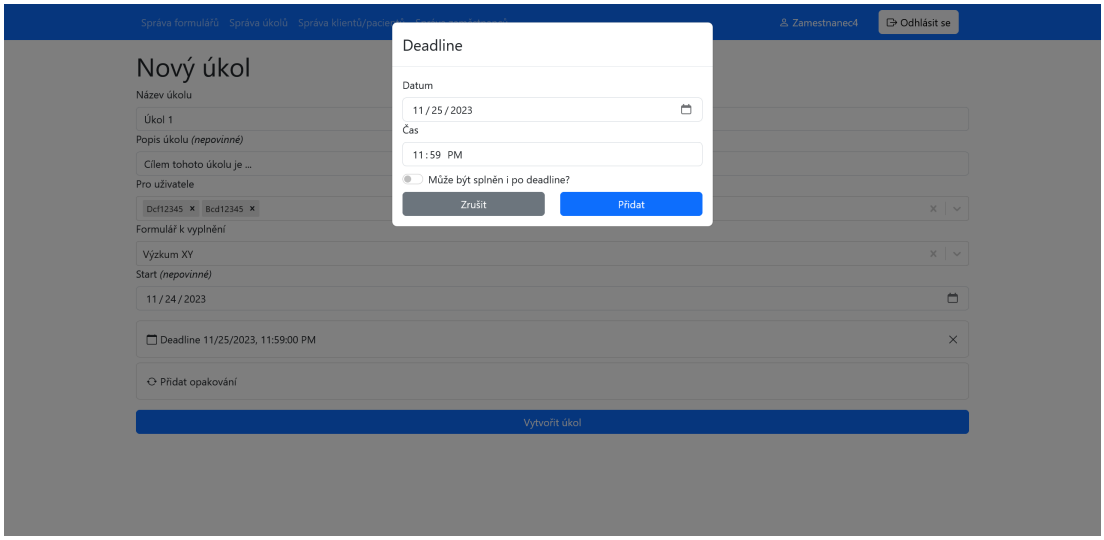
Nyní můžeme vytvořit úkol pro plnitele. Úkoly se vytváří v sekci „Správa úkolů“ (Obr. 7.9). Nový úkol vytvoříme pomocí tlačítka „Nový úkol“. Stisknutím tohoto tlačítka se dostaneme na stránku pro tvorbu úkolu (Obr. 7.10). Pro vytvoření je potřeba zadat název úkolu, vybrat formulář, který má plnitel vyplnit, a vybrat plnitele. Při tvorbě je možno zvolit více plnitelů a tím zadat více úkolů najednou. K úkolu můžeme volitelně přidat popis, start, deadline a opakování. Start úkolu je datum a čas, od kdy je možné úkol splnit. Deadline úkolu je datum a čas, do kdy je možné úkol splnit. Start a deadline byly takto definovány v kapitole 1. Můžeme povolit překročení deadline, ale standardně je po deadline úkol uzavřen a nelze jej splnit. Obrazovka pro konfiguraci deadline je zobrazena na obrázku 7.11. Při nastavení opakování je vytvořeno více úkolu najednou pro jednoho uživatele. Pokud chceme vytvořit opakující se úkol, tak je nutné také definovat deadline, který je vždy posunut o interval specifikovaný v nastavení opakování. Obrazovka pro konfiguraci opakování je zobrazena na obrázku 7.12.



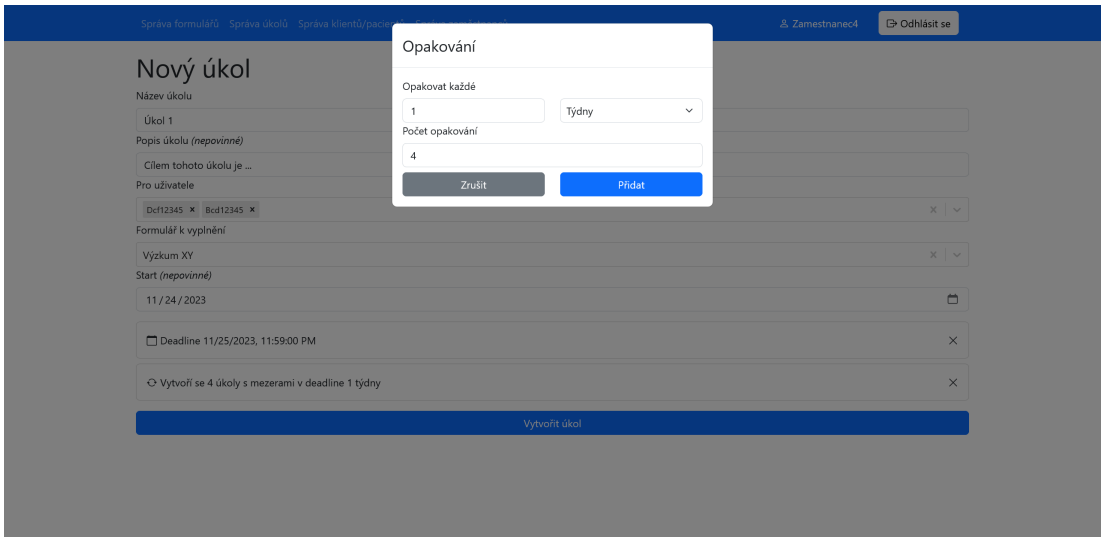
Obrázek 7.9: Správa úkolů



Obrázek 7.10: Tvorba úkolu



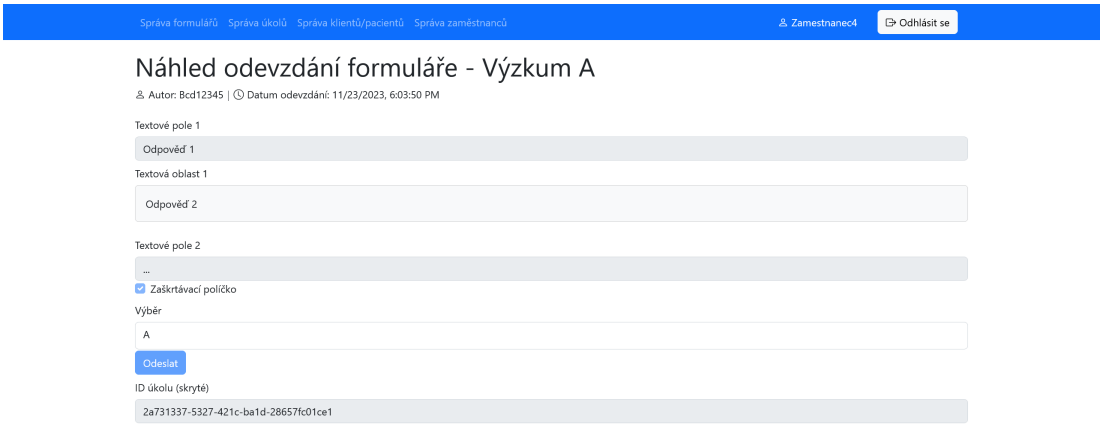
Obrázek 7.11: Zadávání deadline při tvorbě úkolu



Obrázek 7.12: Konfigurace opakování při tvorbě úkolu

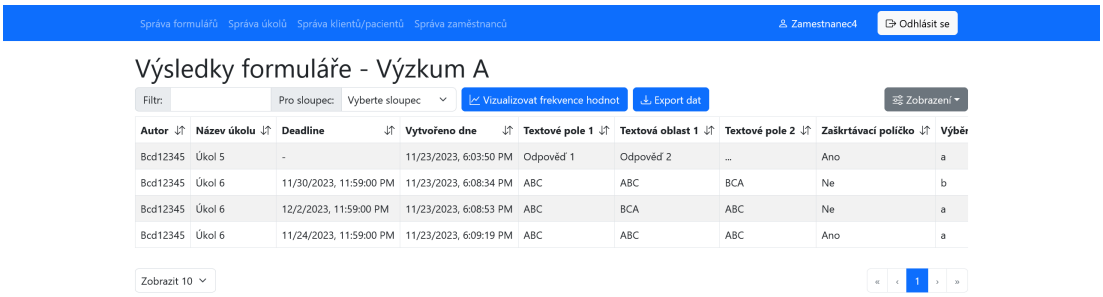
7.2.4 Náhled na odevzdání formuláře

Nyní předpokládejme, že plnitel splnil zadaný úkol. Zaměstnanec si může zobrazit odevzdání formuláře pomocí tlačítka „Zobrazit odevzdání“ v sekci „Správa úkolů“ (Obr. 7.9). Náhled na odevzdání obsahuje metadata vyplnění formuláře a vyplněný formulář (Obr. 7.13). Náhled na odevzdaní zobrazuje i skryté prvky formuláře a jejich hodnoty.

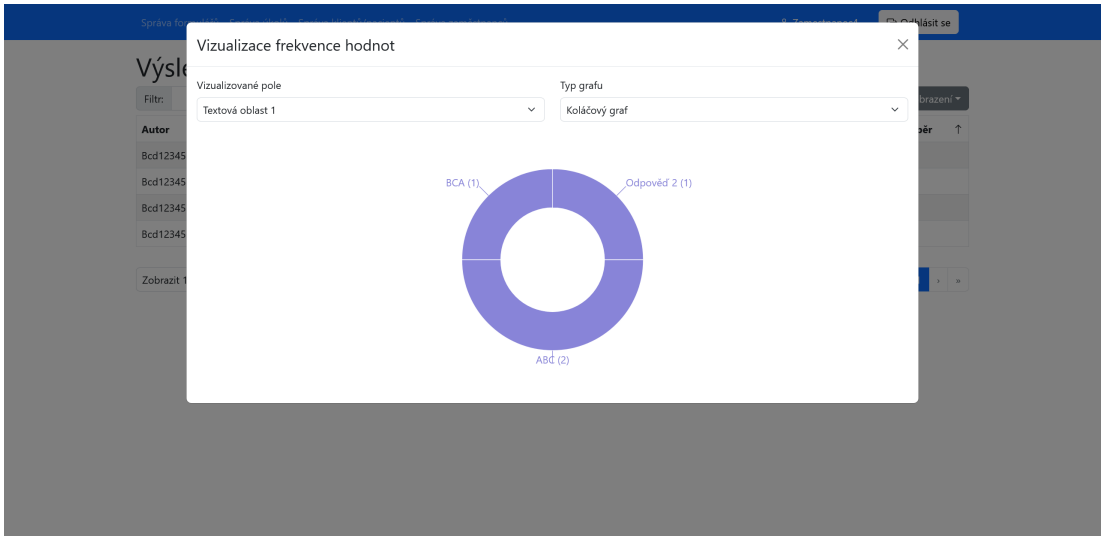


Obrázek 7.13: Náhled na jednotlivé odevzdání formuláře

Můžeme také zobrazit všechna odevzdání formuláře formou tabulky pomocí tlačítka „Zobrazit všechna odevzdání“ v sekci „Správa formulářů“ (Obr. 7.2). Tato stránka obsahuje seznam všech odevzdání formuláře (Obr. 7.14), které lze řadit, filtrovat a následně i exportovat do souboru. Stránka také umožňuje vytvořit základní vizualizace sesbíraných dat, které lze zobrazit stisknutím tlačítka „Vizualizovat frekvence hodnot“ (Obr. 7.15). Pokročilejší analýzy a vizualizace dat je možno provádět specializovaným softwarem, který je schopen zpracovat exportovaná data.



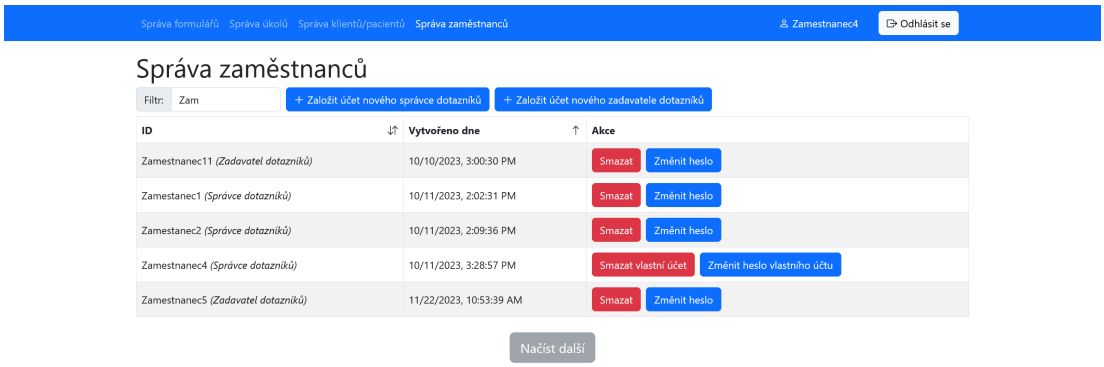
Obrázek 7.14: Náhled na všechna odevzdání formuláře



Obrázek 7.15: Vizualizace odevzdání formuláře

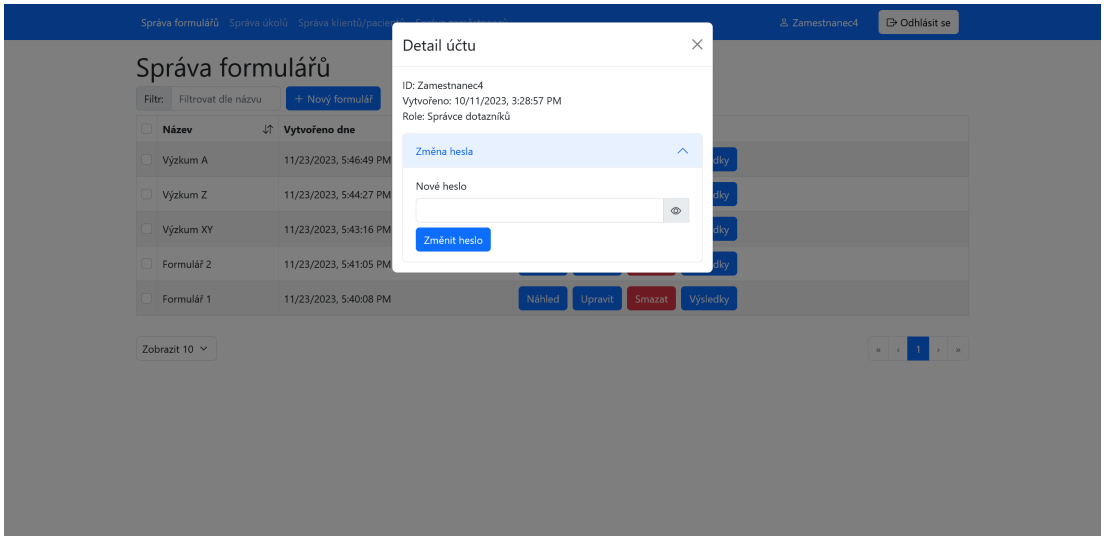
7.2.5 Správa účtů

Zaměstnanec může tvořit účty pro další zaměstnance. Pokud má zaměstnanec roli „Správce dotazníků“, tak může tvořit účty pro další zaměstnance s rolí „Správce dotazníků“ nebo „Zadavatel dotazníků“. Pokud má zaměstnanec roli „Zadavatel dotazníků“, tak může tvořit účty pro další zaměstnance s rolí „Zadavatel dotazníků“. Pohled zaměstnance s rolí „Správce dotazníků“ je zobrazen na obrázku 7.16.



Obrázek 7.16: Správa zaměstnaneckých účtů

Zaměstnanec si může zobrazit detail svého účtu (Obr. 7.17) kliknutím na své ID v pravém horním rohu aplikace. V detailu účtu může zaměstnanec změnit své heslo.

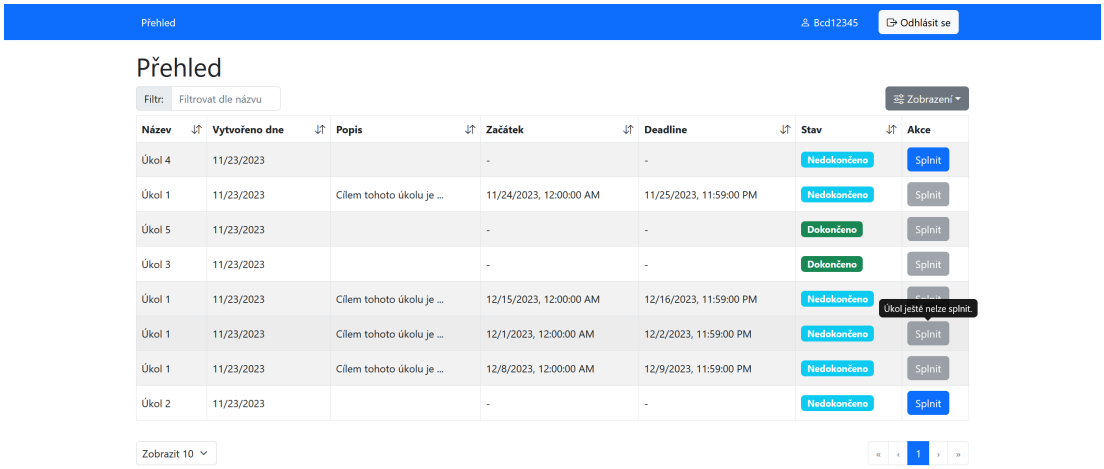


Obrázek 7.17: Změna hesla vlastního účtu

7.3 Užívání aplikace z pohledu plnitele

Nyní popišme proces z pohledu plnitele. Jsou zde popsány veškeré funkce, které byly identifikovány jako požadavky na funkcionalitu dostupnou plniteli v kapitole o analýze požadavků 1.

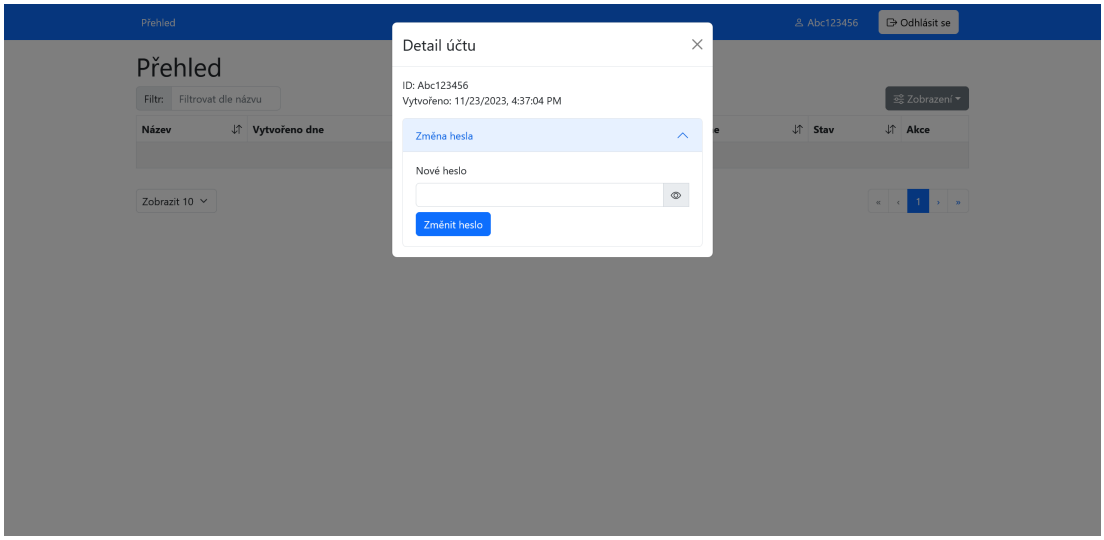
Předpokládáme, že plnitel má již vytvořený účet a přihlásil se do aplikace, jak je popsáno v sekci 7.1, která se věnuje přihlášení a tvorbě účtu. Po přihlášení se plnitel dostane na přehled úkolů (Obr. 7.18). Tato stránka zobrazuje všechny úkoly, které byly plniteli zadány. Plnitel může úkol splnit stisknutím tlačítka „Splnit“, čímž se dostane na stránku obsahující formulář k vyplnění (Obr. 7.19). Plnitel může v průběhu vyplňování formuláře stisknout tlačítko „Uložit koncept“, čímž se formulář uloží do systému, ale neodešle se. Při návratu na tuto stránku je uložený stav automaticky znovu načten.



Obrázek 7.18: Přehled úkolů uživatele

Obrázek 7.19: Vyplnění formuláře plnitelem

Uživatel si může zobrazit detail svého účtu (Obr. 7.20) kliknutím na své ID v pravém horním rohu aplikace. V okně detailu účtu může zaměstnanec změnit své heslo. Pro změnu hesla je nutno zadat nové heslo a stisknout tlačítko „Změnit heslo“. Heslo musí obsahovat alespoň jedno velké písmeno, alespoň jedno malé písmeno a alespoň jedno číslo. Pro manuální kontrolu hesla je možno zobrazit obsah hesla stisknutím tlačítka se symbolem oka.



Obrázek 7.20: Změna hesla vlastního účtu

8. Zhodnocení vývoje

V této kapitole bych se chtěl ohlédnout zpátky, zhodnotit průběh celého vývoje a sepsat získané zkušenosti.

8.1 Zhodnocení architektury

Tento projekt byl vyvíjen jako distribuovaný systém. Toto rozhodnutí pramenilo z mé nezkušenosti a zpětně ho považuji za chybné. Distribuovaná architekura je vhodná pro specifické projekty, které dokáží těžit z výhod, které tato architektura poskytuje. Tento projekt získal pouze minimum výhod, ale musel řešit všechnu komplexitu, která je s distribuovanými systémy spojena.

8.2 Zkušenost s Form.io

Systém spravující formuláře Form.io na mě na začátku působil velmi profesionálně a vyspěle. Narazil jsem však na mnoho problémů a nedostatků, které mi výrazně zkomplikovaly práci. Software má mnoho dokumentace, ale některé části jsou nesrozumitelné a některé funkce jsou nedostatečně zdokumentované. Klientské knihovny mají mj. špatnou podporu TypeScript, nepodporují všechny funkce serveru a mají špatně navržené rozhraní. Po několika měsících práce jsem došel k závěru, že některé klientské knihovny je potřeba kompletně nahradit vlastním řešením.

Dalším problémem byla špatná spolupráce se správci projektu. V průběhu mé práce jsem narazil na nedostatky software Form.io či dokumentace a tyto nedostatky jsem vždy nahlásil, či jsem je sám opravil. Většina z nahlášených chyb dodnes nebyly adresovány a návrhy na mé vylepšení nebyly přijaty.

Kdybych tento projekt začínal znovu, mnohem více bych se snažil zadavateli rozmluvit požadavek na ukládání dat na vlastním serveru. Tento požadavek je totiž hlavním důvodem, proč jsem se nakonec rozhodl pro použití Form.io. Myslím si, že existuje mnoho lepších řešení pro správu formulářů, ale jedná se o cloudové služby. V průběhu vývoje jsem se dozvěděl o HIPAA certifikaci. Software držící tuto certifikaci musí splňovat přísná pravidla pro ukládání zdravotnických dat. Tuto certifikaci má mnoho z nástrojů zmiňovaných v kapitole 2. Myslím si, že užití nástroje s touto certifikací pro tvorbu a správu formulářů by byl nejlepší kompromis mezi bezpečností a kvalitou nástroje. O této možnosti jsme však já ani zadavatel nevěděli a proto jsme se rozhodli pro použití Form.io.

8.3 Zkušenost s NextJS

Rád bych také popsal svou zkušenost s framework NextJS. Framework mi poskytl mnoho skvělých funkcí jako je routing, caching a také sdílení a zanořování layoutů. Framework má skvělou dokumentaci a velkou aktivní komunitu. Narazil jsem však i na několik problémů.

První problém byl s tzv. *hot module replacement*. Tato funkce umožňuje přidat, odebrat nebo vyměnit moduly za běhu aplikace bez nutnosti dalšího načtení

celé stránky [5]. Kompilační systém reaguje na změny souborů, které hlásí souborový systém, a následně aplikuje změny za běhu aplikace. Pokud však vývojový server pustíme v Docker kontejneru, kde uděláme mount složky s kódem z hostitelského systému Windows, tak se změny v souborech nedetekují. Tato chyba v Windows subsystem for Linux (WSL) je známá již od roku 2019 a zatím nebyla opravena (viz diskuze v Github issue).

Dalším problémem byla omezenost middleware, který NextJS poskytuje. Middleware je kód, který je spuštěn před zpracováním všech požadavků na server. Tento kód však běží v speciálním optimalizovaném prostředí nazývané *edge runtime*. Edge runtime je velmi omezené prostředí a spoustu knihoven na něm nelze použít. Důsledky toho jsou, že například nelze použít populární knihovny pro logování (Pino, Winston, apod.) ani knihovnu axios, která poskytuje alternativu k nativnímu fetch API. NextJS v tuto chvíli nedovoluje použití jiného běhové prostředí pro middleware.

Posledním problémem byla implementace *content security policy*. Content security policy je bezpečnostní vrstva, která zabraňuje určitým typům útoků jako je například cross-site scripting (XSS). Pro použití tohoto bezpečnostního prvku je potřeba nakonfigurovat hlavičky HTTP odpovědí a také modifikovat zdrojový kód stránky. Podpora tohoto bezpečnostního mechanismu je v NextJS aktivně vyvíjena, ale v době psaní této práce nebyla plně funkční, což snižuje bezpečnost aplikace.

8.4 Open-source vývoj

Při vývoji jsem narazil na mnoho chyb a nedostatků v použitých nástrojích a knihovnách. Tyto chyby jsem hlásil správcům jednotlivých projektů a některé jsem dokonce sám opravil. V případě některých projektů byla komunikace se správci velmi dobrá a chyby byly rychle opraveny. V některých případě jsem však narazil na rozdílnosti v názorech na správné řešení problémů nebo jsem se nedočkal žádné reakce. Zde je výčet issues a pull requestů, které jsem vytvořil:

- Chyba v internacionalizaci komponenty poskytované knihovnou
- Oprava chyby v internacionalizaci komponenty poskytované knihovnou
- Bezpečnostní chyba v Form.io klientské aplikace
- Chybějící dokumentace API endpointu
- Chyba ve verzi databáze
- Návrh na zlepšení organizace repozitáře
- Použití zastaralého API
- Chyba v pluginu do Docusaurus
- Návrh na vylepšení pluginu do Docusaurus
- Nefunkční příklad v dokumentaci

- Návrh na vylepšení chování statické analýzy kódu
- Chyba v dokumentaci GitHub Action pro upload na Netlify

8.5 Spolupráce s NUDZ

Spolupráce s Národním ústavem duševního zdraví nebyla vždy jednoduchá. Nízká technická znalost zaměstnanců byla větší problém než jsem očekával. V některých chvílích jsem nedokázal klást ty správné otázky a nepodařilo se mi zachytit všechny důležité informace o doméně. Celá spolupráce byla značně ztížena tím, že se v průběhu vývoje několikrát změnila osoba, která se mnou spolupracovala a komunikovala. Celkově jsem však spokojený s výsledkem a věřím, že aplikace bude využívána a pomůže zlepšit kvalitu poskytované péče.

Závěr

V této práci jsme se zabývali vývojem webové aplikace pro monitorování mentálního zdraví. Na začátku jsme detailně analyzovali požadavky zadavatele a vytvořili návrh aplikace. Poté jsme rozebrali proces vývoje, popsali technické detaily a sepsali dokumentaci. Na konci jsme zhodnotili průběh celého vývoje a výslednou aplikaci.

V aplikaci je mnoho možností pro navazující vývoj. Interaktivní vzdělávací hry nebo cvičení by mohly zvýšit motivaci pacientů ke spolupráci. Možnost zasílat zprávy mezi terapeutem a pacientem by bylo možné využít pro řešení akutních potíží, či nejasností v úkolech. Schopnost systému sledovat detailní chování pacienta jako je pohyb myši, čas strávený na konkrétní otázce, apod. v průběhu vyplňování dotazníku by mohlo pomoci při výzkumu.

Výsledkem této práce je plně funkční webová aplikace, která splňuje všechny požadavky zadavatele a je připravena do produkce. Aplikace má mnoho potenciálu pro další rozvoj a využití v praxi.

Seznam použité literatury

[1] What is CRUD? URL https://www.codecademy.com/article/what-is-crud. Navštíveno 6. 4. 2024.

[2] eslint. URL https://www.npmjs.com/package/eslint. Navštíveno 6. 4. 2024.

[3] Data collection. URL https://github.com/formio/formio/issues/1499. Navštíveno 6. 4. 2024.

[4] Camunda + Keycloak + Form.io (CamundaCon 2019). URL https://www.youtube.com/watch?v=nuf46N5vU34. Navštíveno 6. 4. 2024.

[5] Hot Module Replacement. URL https://webpack.js.org/concepts/hot-module-replacement/. Navštíveno 6. 4. 2024.

[6] What Is an In-Memory Database? URL https://aws.amazon.com/nosql/in-memory/. Navštíveno 6. 4. 2024.

[7] jshint. URL https://www.npmjs.com/package/jshint. Navštíveno 6. 4. 2024.

[8] Telemetry. URL https://nextjs.org/telemetry. Navštíveno 6. 4. 2024.

[9] Profil. URL https://www.nudz.cz/o-nas/profil. Navštíveno 6. 4. 2024.

[10] What is an ORM? URL https://www.prisma.io/dataguide/types/relational/what-is-an-orm. Navštíveno 6. 4. 2024.

[11] react-i18next. URL https://www.npmjs.com/package/react-i18next. Navštíveno 6. 4. 2024.

[12] Save as Draft. URL https://help.form.io/faq/tutorials-and-workflows/save-as-draft. Navštíveno 6. 4. 2024.

[13] Autosave your response progress on a Google Form. URL https://support.google.com/docs/answer/10952360?hl=en. Navštíveno 6. 4. 2024.

[14] Save and return to your typeform later. URL https://www.typeform.com/help/a/save-and-return-to-your-typeform-later-360029581051/. Navštíveno 6. 4. 2024.

[15] Managing data consistency in a microservice architecture using Sagas - part 1. URL https://microservices.io/post/microservices/2019/07/09/developing-sagas-part-1.html. Navštíveno 6. 4. 2024.

[16] March 2023 Web Server Survey. URL https://www.netcraft.com/blog/march-2023-web-server-survey/. Navštíveno 6. 4. 2024.

[17] The state of gRPC in the browser. URL https://grpc.io/blog/state-of-grpc-web/. Navštíveno 6. 4. 2024.

[18] OMG Unified Modeling Language (OMG UML). URL https://www.omg.org/spec/UML/2.5.1/PDF. Navštíveno 6. 4. 2024.

[19] What other companies do we share data with? URL https://www.typeform.com/help/a/what-other-companies-do-we-share-data-with-360029617191/. Navštíveno 6. 4. 2024.

[20] How can I track user behaviour on my Typeform? URL https://community.typeform.com/your-typeform-results-32/how-can-i-track-user-behaviour-on-my-typeform-1777. Navštíveno 6. 4. 2024.

[21] What is diagram as code? URL https://docs.eraser.io/docs/diagram-as-code. Navštíveno 6. 4. 2024.

[22] What Is Passwordless Authentication? URL https://auth0.com/blog/what-is-passwordless-authentication/. Navštíveno 6. 4. 2024.

[23] Why Coding Style Matters. URL https://www.smashingmagazine.com/2012/10/why-coding-style-matters/. Navštíveno 6. 4. 2024.

[24] Fowler, M. Continuous Integration. URL https://martinfowler.com/articles/continuousIntegration.html. Navštíveno 6. 4. 2024.

Seznam obrázků

1.1 Doménový model 9

3.1 C4 model - Systémový kontext 19
3.2 C4 model - Kontejnery 20
3.3 C4 model - Komponenta webová aplikace 22
3.4 Diagram nasazení 23
3.5 Logický datový model 28
3.6 Správa nedokončených vyplnění dotazníků 29

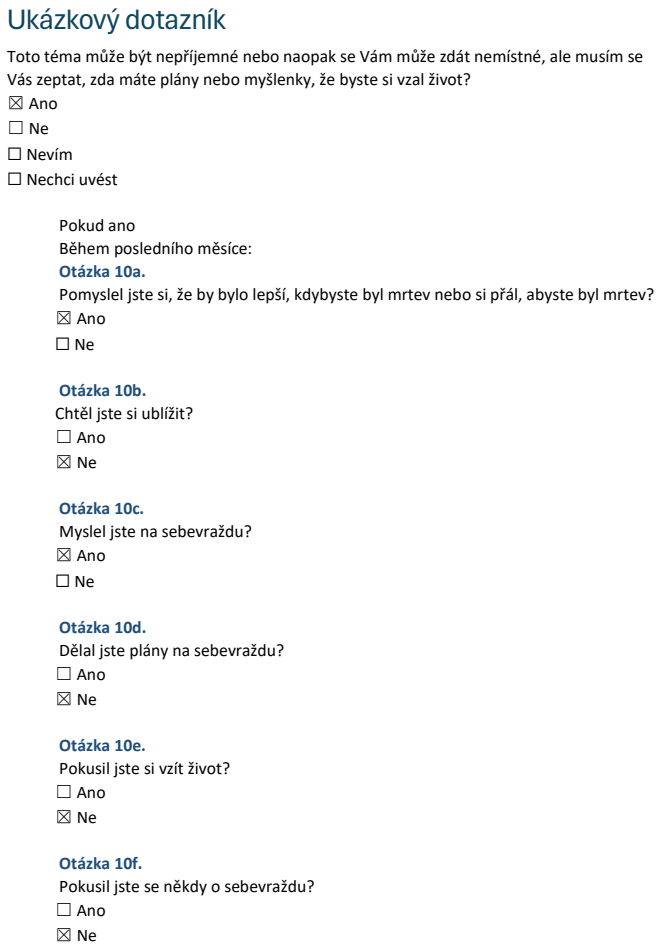
4.1 Komunikace při autentizaci uživatele 33
4.2 Diagram integrace systémů spravující úkoly a systému spravující formuláře 36
4.3 Stavový diagram úkolů 37

7.1 Přihlašovací obrazovka 43
7.2 Správa formulářů aplikace 44
7.3 Tvorba formuláře 45
7.4 Karta Zobrazení v nastavení prvku Skryté 46
7.5 Karta API v nastavení prvku Skryté 46
7.6 Karta Data v nastavení prvku Skryté 47
7.7 Zobrazení menu nastavení prvku 47
7.8 Správa účtů plnitelů 48
7.9 Správa úkolů 49
7.10 Tvorba úkolu 49
7.11 Zadávání deadline při tvorbě úkolu 50
7.12 Konfigurace opakování při tvorbě úkolu 50
7.13 Náhled na jednotlivé odevzdání formuláře 51
7.14 Náhled na všechna odevzdání formuláře 51
7.15 Vizualizace odevzdání formuláře 52
7.16 Správa zaměstnaneckých účtů 52
7.17 Změna hesla vlastního účtu 53
7.18 Přehled úkolů uživatele 53
7.19 Vyplnění formuláře plnitelem 54
7.20 Změna hesla vlastního účtu 54

A.1 Ukázkový dotazník poskytnutý zadavatelem 62
A.2 Přehledová stránka pro definice dotazníků pro zaměstnance 63
A.3 Stránka s výsledky dotazníků pro zaměstnance 63
A.4 Stránka s úkoly pro uživatele 64

A. Přílohy

A.1 Vstupy pro analýzu požadavků



Ukázkový dotazník

Toto téma může být nepříjemné nebo naopak se Vám může zdát nemístné, ale musím se Vás zeptat, zda máte plány nebo myšlenky, že byste si vzal život?

☒ Ano
☐ Ne
☐ Nevím
☐ Nechci uvést

Pokud ano
Během posledního měsíce:

Otázka 10a.
Pomyslel jste si, že by bylo lepší, kdybyste byl mrtev nebo si přál, abyste byl mrtev?
☒ Ano
☐ Ne

Otázka 10b.
Chtěl jste si ublížit?
☐ Ano
☒ Ne

Otázka 10c.
Myslel jste na sebevraždu?
☒ Ano
☐ Ne

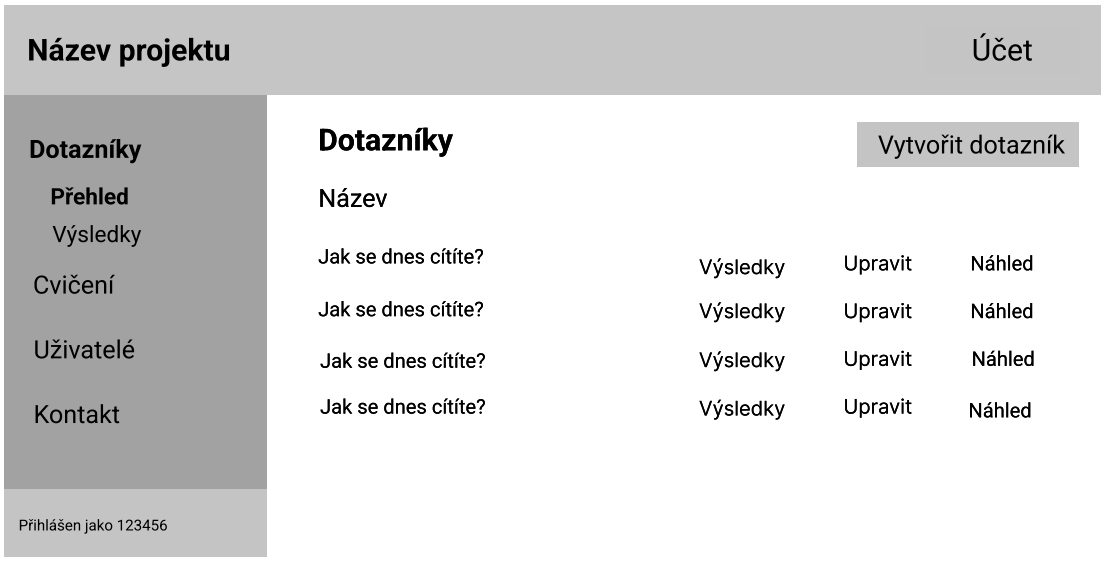
Otázka 10d.
Dělal jste plány na sebevraždu?
☐ Ano
☒ Ne

Otázka 10e.
Pokusil jste si vzít život?
☐ Ano
☒ Ne

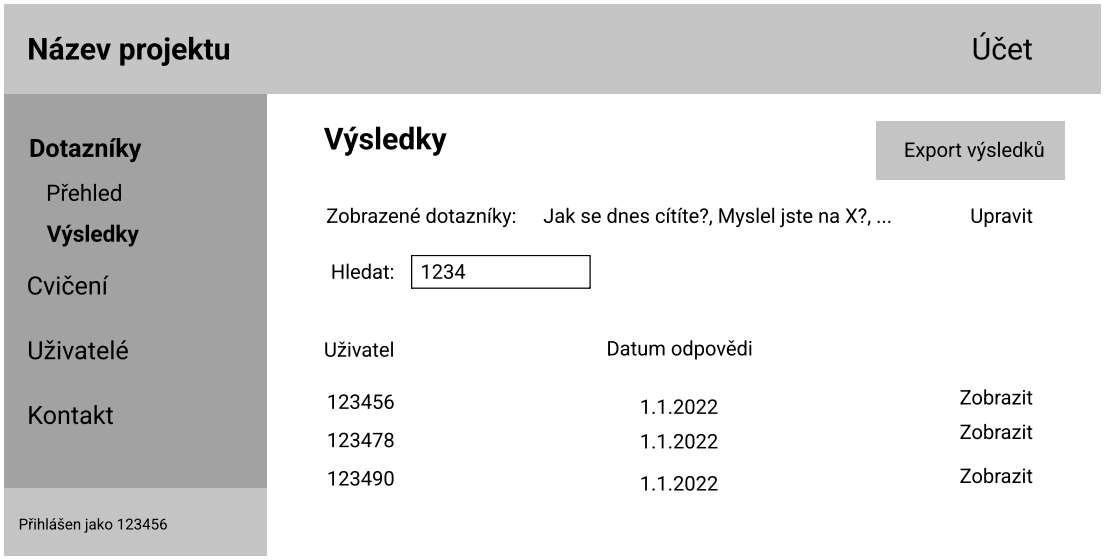
Otázka 10f.
Pokusil jste se někdy o sebevraždu?
☐ Ano
☒ Ne

Obrázek A.1: Ukázkový dotazník poskytnutý zadavatelem

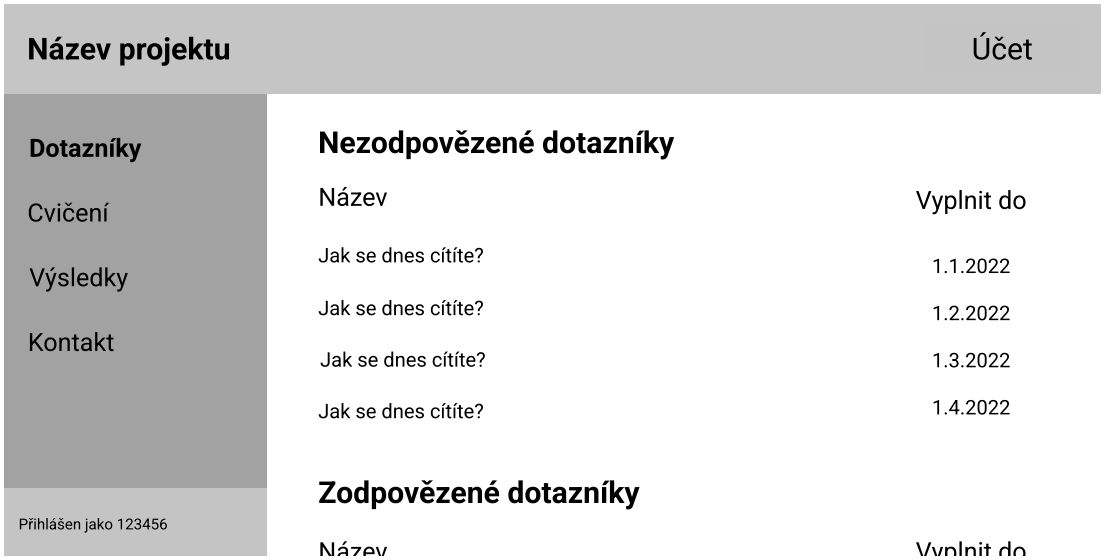
A.2 Wireframy uživatelského rozhraní



Obrázek A.2: Přehledová stránka pro definice dotazníků pro zaměstnance



Obrázek A.3: Stránka s výsledky dotazníků pro zaměstnance



Obrázek A.4: Stránka s úkoly pro uživatele