VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WEBOVÁ A MOBILNÍ APLIKACE PRO ZADÁVÁNÍ A POTVRZOVÁNÍ ÚKOLŮ

WEB AND MOBILE APP FOR ASSIGNING AND CONFIRMING TASKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **FILIP JEŘÁBEK**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Jeřábek Filip**
Program: Informační technologie
Název: **Webová a mobilní aplikace pro zadávání a potvrzování úkolů**
Web and Mobile App for Assigning and Confirming Tasks
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s problematikou zadávání a sledování úkolů dispečerem na stavbách.
2. Prostudujte existující jednoduché online nástroje pro zadávání a sledování úkolů.
3. Navrhněte systém pro zadávání a sledování úkolů dispečerem stavby.
4. Implementujte server a klienty řešeného systému.
5. Testujte systém na uživatelích a iterativně jej zdokonalujte.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Joel Marsh: UX for Beginners: A Crash Course in 100 Short Lessons, O'Reilly 2016

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, značné rozpracování bodů 4 a 5.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Práce řeší řízení pracovníků dispečerem. Cílem je zefektivnit a zjednodušit jak práci dispečera, tak i samotné úkony pracovníků. Zaměřil jsem se na rozdělování úkolů, upozornění na události a změny stavů úkolů, a to vše v reálném čase. Zvolená problematika je řešena pomocí webové aplikace na straně dispečera a mobilní aplikace na straně pracovníků. Webová aplikace je tvořená pomocí PHP frameworku Symfony a mobilní aplikace pomocí frameworku Flutter. Základní myšlenka spočívá ve zjednodušení komunikačního procesu. Dispečer může kdykoliv okamžitě zjistit stav úkolů, které přidělil pracovníkům a pracovník je ihned upozorněn na přidělení úkolu, či změnu jeho stavu. Vytvořené řešení značně zpřehledňuje a zjednodušuje práci dispečera. Díky okamžité odezvě, kterou aplikace poskytuje, se také podařilo eliminovat prostoje a zbytečné cesty pracovníků, čímž se celý proces zefektivňuje. V návaznosti na tyto skutečnosti je předpokládáno zmenšení celkových nákladů a zároveň vykonání většího množství práce za stejný časový úsek.

Abstract

This work solves managing employees by a dispatcher. The goal is to increase the efficiency of the work of a dispatcher, as well as a single employee's tasks. I focused on assigning tasks, event notification, and tasks state changes and all that in real-time. The solution consists of a website application that is used by a dispatcher and mobile application for employees. Website application is made in the PHP framework Symfony and the mobile app is made in the Flutter framework. The whole idea is to simplify the communication process. The dispatcher is able to check the state of assigned tasks whenever he needs, as well as the employee is notified about the newly assigned task, or changed task state, immediately. The solution significantly simplified the dispatcher's job. The idle times and unnecessary employee journeys were eliminated thanks to the immediate response, which made the whole process more effective. Following these facts, the overall costs should be lowered, and also more work should be made in the same amount of time.

Klíčová slova

mobilní, webová, aplikace, úkoly, dispečer, pracovník, potvrzení, přiřazení, upozornění, efektivita, Symfony, CSS, HTML/Twig, Flutter

Keywords

mobile, web, app, tasks, dispatcher, worker, confirmation, assigning, notification, efficiency, Symfony, CSS, HTML/Twig, Flutter

Citace

JEŘÁBEK, Filip. *Webová a mobilní aplikace pro zadávání a potvrzování úkolů.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Webová a mobilní aplikace pro zadávání a potvrzování úkolů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Filip Jeřábek
27. května 2020

Poděkování

Mé poděkování patří panu prof. Ing. Adamu Heroutovi, Ph.D za cenné rady, trpělivost, ochotu a odborný dohled při tvorbě této bakalářské práce. Rád bych také poděkoval Ing. Ctiradu Jeřábkovi za neúprosnou, ale konstruktivní kritiku vedoucí k zdokonalení práce, dále Ing. Aleši Potůčkovi a všem ostatním, kteří se podíleli na testování mé aplikace. A v neposlední řadě pak mé rodině a všem blízkým, kteří mě podporovali při tvorbě práce.

Obsah

Kapitola 1

Úvod

V práci je popsán vývoj mobilní a webové aplikace zabývající se řízením pracovníků v terénu dispečerem. Práce obsahuje úvod do dané problematiky, funkční návrhy i návrhy uživatelského rozhraní, popis implementace a testování aplikací. Webová aplikace je dostupná na adrese mobilnidispecer.cz a mobilní ve službě Obchod play[1].

Dispečer přijímá objednávky a vytváří jednotlivé úkoly. Tyto úkoly pak přiděluje pracovníkům, kteří dané úkoly plní. Pracovník v mobilní aplikaci potvrzuje dokončené úkoly a zároveň má přehled o všech jemu přiřazených (aktuálních či budoucích) úkolech. Dispečer má rychlejší a přehlednější zpětnou vazbu o aktuálních stavech úkolů a může tak lépe rozdělovat další úkoly. Důležitým aspektem aplikace je také funkce upozorňování na blížící se termín úkolu, a to jak pracovníků, tak i dispečera. Aktuálně se ve firmě k rozdělování úkolů využívají převážně telefonní hovory a osobní setkání, což není zcela efektivní.

Pro tvorbu těchto aplikací jsem se rozhodl na základě požadavku stavební firmy. Vytvoření mojí první mobilní aplikace pro mě bylo výzvou. Také jsem se chtěl zdokonalit v tvorbě aplikací s uživatelským rozhraním. Aplikace jsou vyvíjeny dle požadavků konkrétní stavební firmy, která je bude využívat. V rámci vývoje aplikací proběhly konzultace a testování v této firmě. Na základě výsledků byla vyladěna funkčnost mobilní a webové aplikace tak, aby se dosáhlo co možná největšího zefektivnění práce a úspory času i prostředků. Při tvorbě práce byl však kladen důraz na univerzálnost aplikací, aby je bylo možné použít i v jiných oborech, než ve stavebnictví.

Cílem je zjednodušení a zefektivnění komunikačního procesu. Díky aplikacím budou mít jak pracovníci v terénu, tak i dispečer v kanceláři lepší přehled o aktuálním dění. V návaznosti na tyto skutečnosti by se měla zvýšit efektivita práce a současně snížit náklady.

První část práce se věnuje popsání použitých technologií a teorie, o kterou se práce opírá. Je zde také vysvětlen proces práce dispečera a pracovníků. Dále jsou zde popsány podobné již existující aplikace a jejich stručné zhodnocení.

Druhá část práce se soustředí na celkový návrh vytvořených aplikací. Jsou zde shrnuty požadavky na aplikace, zobrazeny možné případy užití, návrh databáze a pak také konkrétní návrhy webové a mobilní aplikace včetně návrhu jejich uživatelského rozhraní. Na závěr této části jsou popsány výhody mého řešení.

Poslední část se zabývá implementací aplikací. Na začátku je popsána tvorba databáze a vzhled aplikací, včetně náhledů obrazovek. Dále obsahuje popis procesu komunikace mezi aplikacemi a také konkrétní implementace vybraných funkcí. Tato část v závěru popisuje i umístění webové aplikace na serveru a zabezpečení mobilní a webové aplikace.

[1] https://play.google.com/store/apps/details?id=com.herokuapp.modis.modil_app5

Kapitola 2

Teorie a zhodnocení současného stavu

2.1 Flutter

Flutter [10] je open-source framework pro mobilní uživatelská rozhraní vytvořený společností Google a vydán v Květnu roku 2017. Umožňuje tvorbu nativních multiplatformních mobilních aplikací s jedním zdrojovým kódem. To znamená, že s jedním programovacím jazykem a zdrojovým kódem je možné vytvořit dvě různé aplikace. Pro Android a pro IOS.

Flutter se skládá ze dvou základních částí

- SDK (Software Development Kit) je kolekce nástrojů pro usnadnění vývoje aplikace. Obsahuje nástroje na kompilaci kódu do nativního strojového kódu.
- Framework (knihovna uživatelského rozhraní založená na widgetech) je kolekce znovu použitelných elementů uživatelského rozhraní (tlačítka, textové pole, posuvníky), které je možné upravit dle vlastních potřeb.

Při vývoji flutteru se používá programovací jazyk Dart. Tento jazyk byl vytvořen společnosti Google v Říjnu roku 2011, který se ale od té doby hodně vylepšil. Dart se soustředí na front-end vývoj a dá se použít pro tvorbu mobilních a webových aplikací. Je to typovaný objektově orientovaný programovací jazyk. Jeho syntaxe se dá srovnat s JavaScriptem.

2.1.1 Důvod zvolení Flutteru

Tato práce je moje první zkušenost s vývojem mobilní aplikace. Při hledání vhodných jazyků či frameworků, ve kterých budu svoji aplikaci vyvíjet, jsem narazil na React Native[1] a Flutter[2]. Oba tyto frameworky jsou výbornou volbou pro tvorbu mojí aplikace a jeden oproti druhému nemají žádné zásadní výhody či nevýhody. Vzhledem k mým minimálním zkušenostem s vývojem mobilních aplikací je velkou výhodou, že je snadné se je naučit. Oba frameworky také podorují tvorbu multiplatformní aplikace. V této práci se soustředím pouze na aplikaci pro operační systém Android, ale v dalším budoucím vývoji je možné rozšíření aplikace na mobilní zařízení Apple. React Native byl veřejně vydán v roce 2015. Oproti frameworku Flutter, který byl vydán o dva roky později, v roce 2017, má tedy React výhodu, že je lépe otestovaný a zažitý. Uživatelská základna Flutteru se opravdu rychle

[1] **React Native** – reactnative.dev
[2] **Flutter** – flutter.dev

rozrostla a podpora je velmi obsáhlá. Například v dnešní době má Flutter více hvězdiček (což se dá považovat za alternativní hodnocení) na Githubu[3], než React Native[4]. Flutter má 90,9 tisíc a React Native má 86,5 tisíc. Pro další informace o porovnání těchto dvou frameworků je možné navštívit webovou stránku s porovnáním[5].

Z hlediska statistik má Flutter mnohem větší potenciál, proto jsem se rozhodl ho rozhodl použít pro vývoj aplikace.

2.1.2 Kurz Flutteru

S Flutterem jsem neměl žádné předchozí zkušenosti, a proto jsem se rozhodl najít a projít si vhodný kurz. Hlavní požadavky na kurz byly, aby aplikace byla robustní a komponenty byly použity správně. Další důležitou podmínkou pro kurz bylo, aby učitel dokázal popsat a odůvodnit použití příslušných komponent aplikace.

Prošel jsem několik kurzů, než jsem našel kurz, z kterého bylo jasné, že učitel je odborníkem v praxi a přesně ví, co dělá. Nakonec jsem našel kurz Flutteru [5], který mi naprosto vyhovoval. Tento kurz má dohromady přibližně 30 hodin a je rozdělen do velkého množství kapitol a podkapitol. Učitel do procesu tvorby vzorové aplikace zahrnul i extrémy, které v jeho případě nenastaly, ale při tvorbě jiné aplikace by mohly nastat. Také vše řádně zdůvodňoval a představoval různá alternativní řešení, přičemž vždy řekl pozitiva či negativa a vybral nevhodnější řešení. Kurz stojí přibližně 500 Kč.

Při procházení kurzu jsem nevytvářel přímo vzorovou aplikaci, ale přizpůsoboval ji mému požadovanému řešení do mé bakalářské práce. Tím, že jsem aplikaci tvořil při kurzu, se eliminovalo následné znovu procházení kurzu a vyhledávání představených řešení. To mi pomohlo urychlit vývoj aplikace.

2.2 Symfony

Symfony [8] je kompletní framework pro PHP, tvořený sadou PHP komponent, která výrazně zjednodušuje tvorbu webových aplikací. Framework je kvalitně objektově navržený a světově je velmi rozšířený. Fungují na něm velké i malé projekty. Také stojí za zmínku, že některé jeho komponenty využívají i další nástroje, například Doctrine, Composer, Codeception nebo dokonce redakční systémy jako Drupal, Joomla nebo PrestaShop. Symfony pracuje s architekturou MVC.

Vybrané projekty používající Symfony:

- Laravel
- Google APIs Client Library
- Wikimedia
- phpMyAdmin
- PrestaShop

[3] **Flutter na githubu** – github.com/flutter/flutter
[4] **React Native na githubu** – github.com/facebook/react-native
[5] https://hackr.io/blog/react-native-vs-flutter

2.2.1 MVC

Aplikace řídící se architekturou MVC [8] se skládá ze třech základních typů, které se v aplikaci dělí o 3 základní úlohy:

- **Logiku** – Modely (Models). Obsahují logiku aplikace, jako například práci s databází nebo výpočty. Každá datová entita má většinou svůj model (uživatel, článek, komentář,...).
- **Výstup** – Pohledy (Views). Obsahují Twig šablony s HTML kódem. Twig je šablonovací systém, který do HTML šablon umožňuje vkládat data z PHP pomocí speciálních značek.
- **Řízení** – Kontrolery (Controllers). Kontroler je část, se kterou komunikuje uživatel. Předá jí parametry a ona mu vrátí HTML stránku. Kontroler typicky parametry předá modelům, od kterých získá data. Získaná data předá pohledům (šablonám), které data začlení do HTML kódu. Tento HTML kód pošle kontroler uživateli do prohlížeče. Funguje tedy jako takový prostředník.

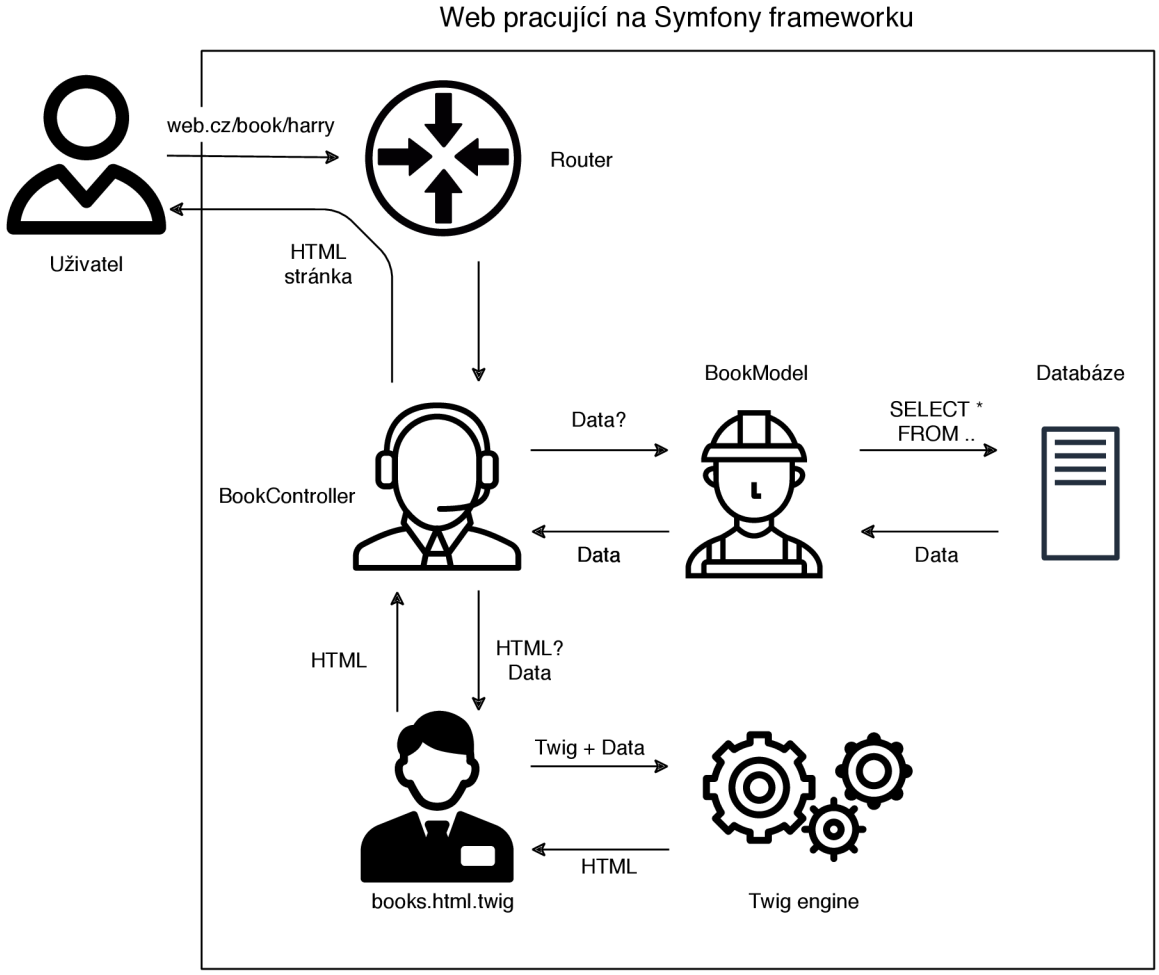
2.2.2 Životní cyklus aplikace v Symfony

Jak už bylo zmíněno, Symfony funguje na principu MVC architektury. Na obrázku 2.1 (převzato z [8]) je diagram životního cyklu Symfony.

1. Jako první se požadavek dostane k routování. Zde se podle adresy zjistí, že uživatel chce něco s knihami, a proto se zavolá BookController a předá se mu zbytek URL.
2. BookController se podívá do parametrů, co se po něm chce a zjistí, že uživatel chce vypsat knihy s názvem „Harry Potter“. Získá si tedy BookModel, kterému sdělí, že chce tyto knihy. BookController vykonává podle dané URL příslušnou metodu třídy, v tomto případě zobrazení seznamu knih. Stejně tak ale může např. knihu přidat nebo odstranit pomocí jiných v něm obsažených metod.
3. BookModel dostane v parametru název knih, které získá z databáze a vrátí.
4. BookController získá data od modelu a tato data předá pohledu (šabloně).
5. Šablona obsahuje HTML stránku pro seznam knih a v ní nějaké Twig značky, do kterých se tato data automaticky doplní. Vložení těchto dat obstará automaticky Twig engine.
6. BookController dostane zpět ze šablony výsledné HTML a to pošle uživateli jako odpověď.
7. Uživateli se v prohlížeči zobrazí HTML stránka.

2.3 Tvorba uživatelského rozhraní

Důležitým aspektem uživatelského rozhraní je **použitelnost**, o které mluví Steve Krug ve své knize [3]. Základní pravidlo, které uvádí jako první je, že nic důležitého by nemělo být dále, než dvě kliknutí. Pokud bychom měli vybrat jedno jediné pravidlo, kterým se budeme řídit, pak je to právě tohle. Z rozhraní by mělo být jasné, jaký je jeho účel. V případě, že



Obrázek 2.1: **MVC.** Na obrázku je životní cyklus Symfony používající vzor Model View Controller.

rozhraní potřebuje vysvětlení, pak je něco špatně. Hlavní myšlenkou knihy (jak napovídá její název) je, že bychom uživatele neměli nutit přemýšlet. To se váže k předchozímu – mělo by být na první pohled jasné, co k čemu slouží. Pro tlačítka bychom měli volit jasné a krátké názvy. Pokud chceme například udělat tlačítko pro zobrazení dostupných prací a pojmenujeme ho „Práce“, pak uživatel okamžitě ví, kam ho tlačítko přesměruje. Můžeme ho pojmenovat například „Pracovní příležitosti“ – tento název má v podstatě stejný význam, ale nutí uživatele se na chvíli zamyslet. Podobně, pokud je nějaký prvek tlačítko, tak by to jako tlačítko měl vypadat. Uživatel je zvyklý na nějakou podobu tlačítka, pokud ji dodržíme, pak je uživateli hned jasné, o co se jedná. Pokud bychom zašli do extrému a změnili nevhodně design, pak uživatel vůbec nemusí poznat, že se jedná o tlačítko.

Na druhou stranu ne vše, co uživatel v našem rozhraní uvidí, už někdy vidět musel. Pokud existuje nějaká funkcionalita, u které není na první pohled jasné co dělá, je nutné uživateli poskytnout prostředky, dle kterých po kratším zamyšlení pochopí, o co se jedná. Je možné zde například vložit krátký popisek či vysvětlivku.

Další důležitá věc, kterou je třeba si uvědomit je, že uživatel nikdy nebude používat webovou stránku tak, jak programátor očekává. Pravděpodobně zprvu proběhne několik chaotických kliknutí, pak návratů na předchozí stránku. Uživatelovu pozornost získají věci,

které nejvíce vystupují. Neexistuje uživatel, který by si stránku plnou textu přečetl shora až dolů, slovo od slova a poté se až začal rozmýšlet, na co klikne. Proto se stránky musí dělat co nejjednodušší a ideálně bez zbytečných a rušivých elementů.

Rozhraní by mělo mít nějaký určitý styl, který bude všude dodržovat. Musí působit jednotným dojmem. Prvky s podobnou funkcionalitou by vždy měly mít podobný vzhled. Je velice zmatečné a náročné hledat na každé podobné stránce tlačítko jiného vzhledu a umístění.

2.3.1 User experience

Česky se dá význam zkratky „User's experience" přeložit jako „Uživatelský zážitek". Zkratka tohoto slova je **„UX"**. Ta se často zaměňuje s UI (User interface – Uživatelské rozhraní). Leč tyto dvě věci spolu úzce souvisí, nejsou jedno a to samé. Jak je popsáno v knize [7]. UX je součástí všeho. Úloha programátora není vytvořit UX, ale zajistit, aby byl uživatelský zážitek dobrý. Častá mýlka je, že UX by mělo udělat uživatele šťastného. To ale není pravda. Pokud by se programátor řídil pouze tímto pravidlem, pak by stačilo, aby uživateli zobrazil nějaký vtipný obrázek či složil kompliment a bylo by hotovo. Pravým cílem UX je, aby uživatel pracoval efektivně. Výkladem této zkratky také není pouze uživatelský zážitek, ale je tím myšlen celý proces tvorby. Programátor musí dostávat zpětnou vazbu od uživatelů, snažit se pochopit myšlení těchto uživatelů a vymýšlet nové a lepší řešení na základě zpětné vazby.

Toto je základní představení UX. Jedná se jen o špičku ledovce jeho teorie. Více informací je možné získat v knize, kterou napsal Joel Marsh [7].

2.3.2 Zásady pro mobilní uživatelské rozhraní

Pro spoustu uživatelů je mobilní telefon rychlejší a pohodlnější způsob, než používat počítač. V dnešní době je možné říci, že mobilní telefony jsou v určitých věcech lepším nástrojem než počítače. Rapidně se zvyšuje jejich výkon, takže je na nich možné dělat operace jako na počítačích. Jejich hlavní výhodou je ale to, že jsou stále zapnuté a uživatelé je stále nosí při sobě. Mezi počítačem a mobilním telefonem jsou však obrovské rozdíly a to hlavně při tvorbě uživatelského rozhraní. Jonathan Stark ve svém článku [9] shrnul nejzákladnější principy tvorby mobilního uživatelského rozhraní. Níže je výpis některých částí těchto principů.

Soustředěnost – více není lépe. **Rozvaha** – je důležité před začátkem tvorby aplikace si představit z uživatelského hlediska, co uživatelé opravdu chtějí a potřebují. **Schopnost reagovat** – je důležité, aby aplikace reagovala na akce uživatele. Rychlost a schopnost reagovat nejsou dvě stejné věci. Některé operace mohou trvat delší dobu, ale uživatel musí dostávat zpětnou vazbu o jejich průběhu. **Uhlazenost** – detaily jsou důležité. Aplikace může obsahovat neskutečné funkcionality, ale pokud nebudou dotažené detaily, tak celá aplikace bude vytvářet špatný dojem. **Prostor na kliknutí** – je důležité, aby objekty, na které uživatel bude klikat, byly dostatečně velké a nepůsobilo problémy se trefit přesně. **Dosah palce** – pokud uživatel nebude k ovládání telefonu používat obě ruce, pak pravděpodobně bude telefon ovládat pouze palcem. Musí dosáhnout. **Modální upozornění** – jsou extrémně otravné při uživatelské práci, jejich využití je oprávněné pouze v nejnutnějších případech. Například pokud je opravdu potřeba, aby uživatel zanechal práce a přečetl si obsah upozornění. **Potvrzování** – potvrzování pomocí modálních oken je více uživatelsky přívětivé. Uživatel opět vidí pouze modální okno, ale to je výsledkem reakce na jeho akci. Pokud jsou tlačítka pro potvrzení pod sebou, vždy je dobré dát tu nejbezpečnější možnost dolů. **Znovu-otevření** – pokud uživatel používal aplikaci a minimalizuje ji, pak by se měl

dostat do stavu, kde byl předtím. **Důležité jsou první dojmy** – ikona aplikace a spouštěcí obrazovka je první, co uživatel aplikace uvidí.

2.4 Firebase Cloud Messaging

FCM (Firebase Cloud Messaging)[6] je multiplatformní služba sloužící pro **notifikace klientských zařízení**. Může se jednat přímo o zprávy zobrazené uživateli či upozornění aplikace na nová data a vyžádání jejich stažení.

Odesílání notifikací pomocí této služby je možné buď všem přihlášeným zařízením, skupinám zařízení (například zařízením přihlášeným k odběru novinek o určitém tématu) nebo jednotlivým zařízením pomocí jejich jednoznačného identifikátoru – tokenu. Token klientské zařízení obdrží po konfiguraci této služby v aplikaci.

2.5 Stručný popis použitých webových technologií

Tato kapitola je stručným průvodcem základů použitých technologií při vývoji webové aplikace. Popisy těchto technologií vyplývají z mých osobních znalostí, doplněných popisem z knih [1], [2], [6] a [12].

2.5.1 HTML5

Význam zkratky je „Hyper Text Markup Language“. Obecně je HTML [2] značkovací jazyk, který slouží k vytváření **webových stránek**. HTML5 je následníkem HTML 4.01. Obsahuje nové funkce, odstraňuje zastaralé, či případně mění jejich chování.

Základní stavební strukturou HTML jsou **tagy**. Jsou to klíčová slova označení elementů uzavřené ve špičatých závorkách `<>`. Tyto tagy se dělí na **párové** (mají začátek `<div>` a konec `</div>`) a **nepárové** (`<br>`). Tagy se do sebe zanořují a tím vzniká struktura HTML dokumentu.

2.5.2 CSS3

„Cascading Style Sheets“ v překladu **kaskádové styly** je jazyk pro popis způsobu zobrazení elementů v jazyce HTML. Tento jazyk se zpravidla nachází ve zvláštním souboru s příponou `.css`, který je vázaný k HTML dokumentu pomocí odkazu v hlavičce HTML. CSS3 [2] je nová verze jazyka CSS 2.1 podobně jako HTML5 z HTML.

2.5.3 Bootstrap

Je to sada nástrojů sloužící pro tvorbu webových aplikací. Je to v podstatě předdefinovaná sada tříd **CSS**, doplněných o kód jazyka JavaScript, které se jednoduše dají přiřadit elementům v HTML dokumentu.

2.5.4 JavaScript

Je to událostmi řízený scriptovací jazyk [12]. V práci je použit jako součást webové aplikace pro asynchronní provádění **operací na pozadí**. Díky čemuž je možné provádět operace, aniž by uživatel musel být přesměrován na jinou stránku aplikace.

[6] https://firebase.google.com/docs/cloud-messaging

2.5.5 PHP

Význam zkratky PHP [6] [11] je „Hypertext Preprocessor“. Původně zkratka zastávala „Personal Home Page“. Časem se ale PHP velice rozšířilo a byl změněn i význam jeho zkratky. Ta se odkazuje na Hypertext z HTML (Hyper Text Markup Language) a na fakt, že kód PHP je prováděn na serveru před odesláním HTML stránky klientovi. Jedná se tedy o dynamicky typovaný jazyk sloužící především pro tvorbu dynamických webových stránek. Jak už bylo řečeno, kód se provádí na straně serveru a klient obdrží pouze vygenerovaný HTML dokument.

V práci je v jazyku PHP implementována převážná část webové aplikace. Framework Symfony je totiž založený na PHP.

2.6 Popis procesu práce dispečera a pracovníků v praxi

Dispečer je nadřízený pracovníků.

Dispečer je zaměstnanec, který pracuje převážně z kanceláře a má neustálý přístup k počítači. Jeho náplní práce je vyhledávat a přijímat zakázky od zákazníků. Po ujasnění a potvrzení zakázky přichází na řadu rozdělování úkolů pracovníkům.

Pracovník je zaměstnanec, který pracuje v terénu. Může se jednat například o řidiče kamionu, zedníka či operátora strojů na stavbě. Tento pracovník přijímá a vykonává úkoly zadané dispečerem.

V současné době se ve firmě, která aplikaci testovala, k rozdělování úkolů pracovníkům využívá převážně hovor přes mobilní telefon či osobní setkání, což je vysoce neefektivní a náročné. Dispečer nemá okamžitý přehled o stavu přidělených úkolů a v případě, že chce zjistit daný stav úkolů, musí je zjišťovat od pracovníků jednotlivě. Dispečer úkoly a zakázky zaznamenává pouze do poznámkového bloku.

2.7 Dostupné aplikace

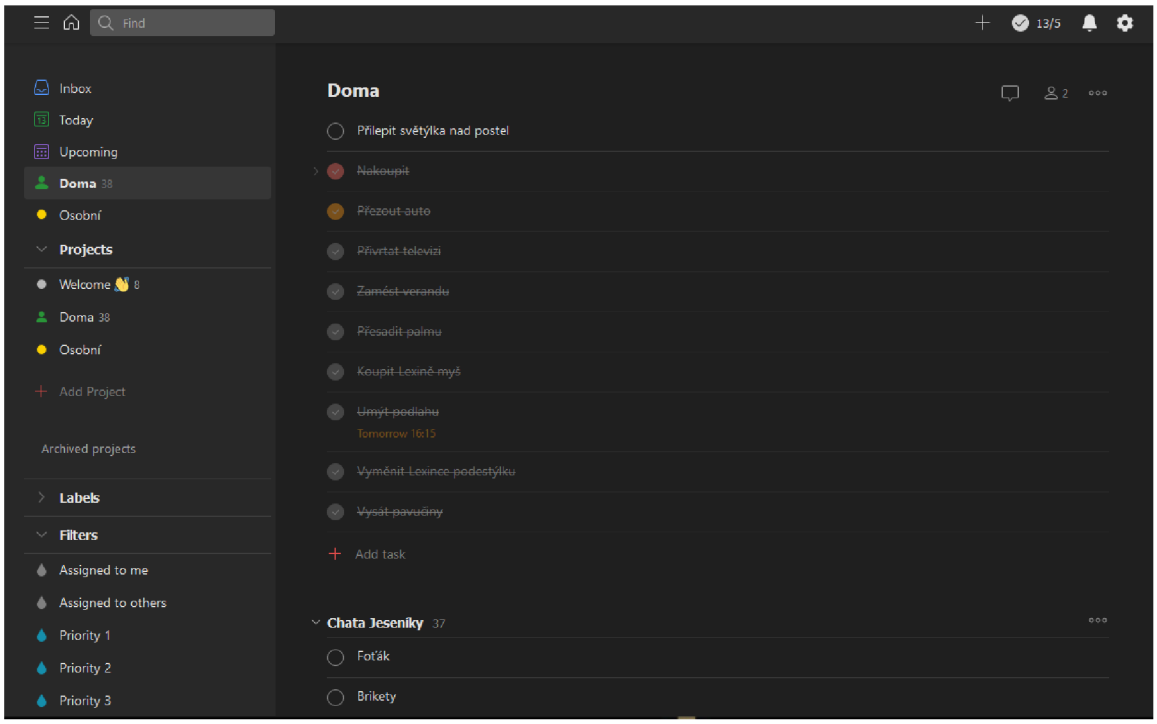
Z dostupných aplikací na internetu neexistuje alternativa k mobilní a webové aplikaci, které jsou předmětem tvorby této bakalářské práce. Úkolové či připomínkové aplikace existují, ale tyto aplikace jsou většinou cíleny na práci pouze jednoho uživatele. Některé z těchto aplikací nabízejí sdílení úkolů mezi více uživateli a zařízeními. Tyto aplikace jsou však pouhé přehledy úkolů a neobsahují logiku návaznosti úkolů, vztah mezi dispečerem a pracovníkem a notifikace přidělení nového úkolu, případně upozornění na úkol blížící se požadovanému termínu dokončení.

2.7.1 Todoist

Todoist[7] je webová (obrázek 2.2) i mobilní (obrázek 2.3) aplikace. V tomto případě se funkčnost mobilní od webové aplikace neliší. Úkoly je zde možno naplánovat na určité datum a čas a aplikace poté úkol připomene. Úkoly mohou mít různou prioritu, podle které se dají filtrovat. Uživatelé mohou zobrazit přehled úkolů na aktuální den či na následujících sedm dnů. Úkoly se zde dají řadit do projektů a tyto projekty se poté dají sdílet s ostatními uživateli.

Jedná se ale pouze o seznamy úkolů a aplikace postrádá některé požadované aspekty, například návaznost úkolů, šablony úkolů a notifikace příchozího úkolu.

[7] Todoist – Aplikace dostupná z todoist.com



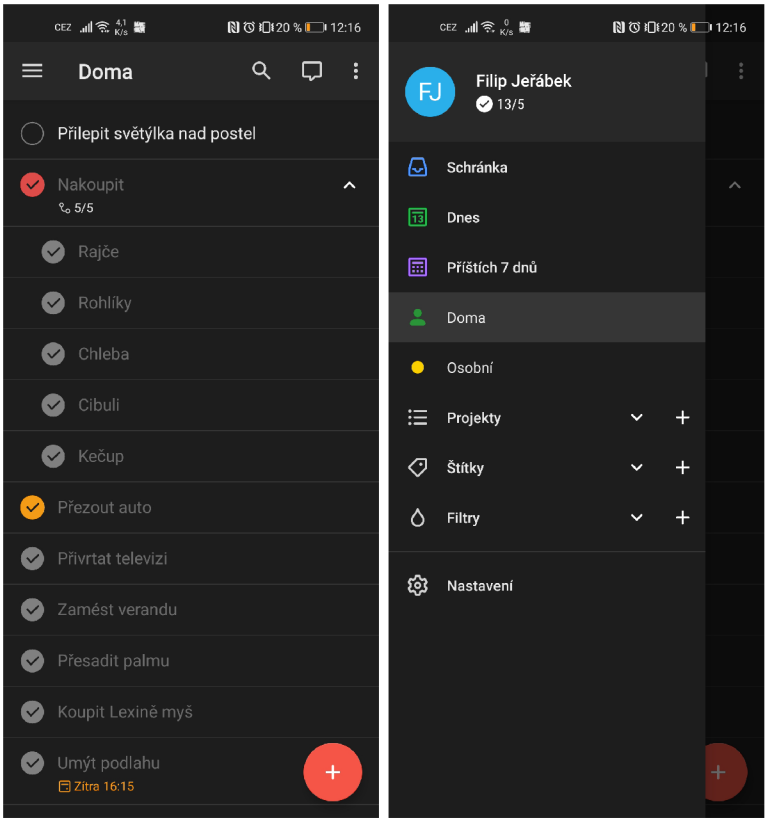
Obrázek 2.2: **Todoist webová aplikace** – Na obrázku je webová verze aplikace Todoist. V levé části obrazovky je menu a filtry. V pravé části obrazovky jsou skupiny úkolů a jednotlivé úkoly. Barevné označení úkolů je pro odlišení priorit.

2.7.2 Fusio

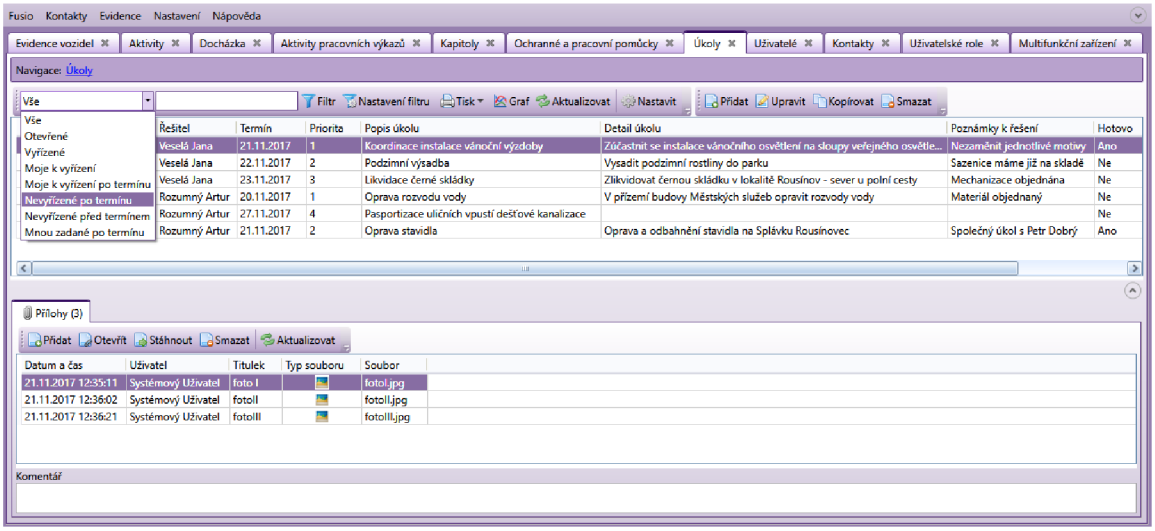
Aplikace Fusio[8] je velice komplexní informační systém. Celá aplikace obsahuje kromě správy úkolů například evidenci vozidel, docházku nebo aktivity. Co se týče sekce s úkoly (obrázek 2.4), tak zde je možné úkolům přiřadit termín a připojit přílohy. Součástí jsou také grafy. V přehledu úkolů jsou k dispozici filtry a seznam příloh.

Aplikace se pro řešený problém nehodí. Funkčnost se dle webové stránky dá individuálně přizpůsobit. Množství klíčových funkcí, které aplikace neposkytuje, je ale pravděpodobně příliš veliké na to, aby se vyplatilo aplikaci přizpůsobit. Úkoly zde na sebe nemohou být navázány a aplikace neobsahuje šablony pro zrychlené zadávání úkolů. Největším nedostatkem je absence mobilní aplikace.

[8] Fusio – Aplikace dostupná z `fusio.cz/ukolovnik/`



Obrázek 2.3: **Todoist mobilní aplikace** – Na obrázku je mobilní verze aplikace Todoist. Na levém obrázku je seznam skupin úkolů a jednotlivých úkolů. Barevné označení úkolů slouží pro odlišení priority. Na pravém obrázku je menu aplikace.



Obrázek 2.4: **Fusio** – Na obrázku je aplikace informačního systému Fusio. Je zobrazena záložka „Úkoly“, která se zabývá problematikou zadávání úkolů. V horní části obrazovky jsou jednotlivé úkoly a ve spodní části jsou přílohy k úkolu.

Kapitola 3

Návrh aplikací

Na základě požadavků, parametrů a ilustračního návrhu na aplikaci, jsem některé aspekty aplikace zjednodušil a vytvořil návrh dvou samostatných aplikací. Mobilní aplikaci pro pracovníky, jímž jsou úkoly zadávány, a webovou aplikaci pro dispečera, který úkoly pracovníkům zadává. Jak tyto návrhy, tak i prototypy a částečná řešení jsem neustále konzultoval s firmou, která aplikaci testuje, pro dosažení co nejlepších výsledků. Kvůli plánované univerzálnosti aplikace jsem musel od některých požadavků firmy opustit, ale navrhl však jsem vhodné alternativy, které splnily původní požadovanou funkcionalitu.

3.1 Požadavky na aplikace

Základní princip práce je přidělování úkolů pracovníkům dispečerem. Dispečer na základě objednávek tvoří úkoly a ty poté rozděluje pracovníkům. Informace, které úkol obsahuje jsou:

- Číslo úkolu
- Název
- Popis
- Uživatel, kterému je úkol přiřazen
- Stav úkolu
- Termín požadovaného dokončení
- Lokace

Práce se dělí do dvou základních částí. Těmito částmi jsou mobilní aplikace a webová aplikace.

Mobilní aplikaci budou používat pracovníci plnící přiřazené úkoly. Pracovník zde bude mít přehled o všech jemu přiřazených úkolech. Pomocí mobilní aplikace musí být schopen potvrdit jejich dokončení, případně zrušit potvrzení o dokončení. Aplikace pracovníka musí informovat o nově příchozím úkolu a o úkolu, který se blíží k termínu požadovaného dokončení. Aplikace by měla bez nutnosti zásahu pracovníka udržovat seznam úkolů aktuální, aby nedocházelo k případům, kdy pracovník kontroluje jemu přiřazené úkoly, ale vidí pouze starý, neaktualizovaný seznam jeho úkolů.

Webovou aplikaci bude využívat dispečer, který bude řídit činnost pracovníků tím, že jim bude přidělovat úkoly. Dispečer je osoba, která má přístup k počítači po celou pracovní dobu a jeho náplní práce je aktivní vyřizování objednávek a řízení pracovníků. V aplikaci dispečer vytvoří úkol, jemuž zadá příslušné parametry a tento úkol odešle pracovníkovi. Rozhraní dispečerovi poskytne přehled všech jemu podřízených pracovníků a jejich přidělených úkolů. Uživatel bude mít také možnost úkol vytvořit, uložit a ponechat v aplikaci na pozdější přiřazení. Aplikace musí poskytovat prostředky pro vazby mezi úkoly. Úkoly mohou být propojovány pro určení vztahu následujícího a předcházejícího úkolu. Aplikace musí také dispečera upozornit na úkol blížící se termínu dokončení.

Speciálním požadavkem pro webovou aplikaci využívanou dispečerem je potřeba šablon úkolů. Tyto šablony si dispečer v aplikaci bude moci vytvořit a s jejich pomocí může efektivněji vytvářet a přidělovat úkoly s podobnými parametry.

3.2 Případy užití

Diagram případů užití na obrázku 3.1 zobrazuje možná použití aplikací uživateli. Typy uživatelů jsou tři. Administrátor webové aplikace, dispečer používající webovou aplikaci a pracovník využívající mobilní aplikaci. Všichni uživatelé se před prací budou muset přihlásit do příslušné aplikace. Aplikační účty jsou odděleny. To znamená, že se pracovník nemůže přihlásit do webové aplikace a dispečer se nemůže přihlásit do mobilní aplikace.

Administrátor je globální role v celé webové aplikaci. Jeho hlavní činností bude přidávání firem a dispečerů do aplikace. Aplikace administrátorovi bude poskytovat veškeré funkcionality jako dispečerovi, rozšířené o administrátorské záležitosti, jako je právě správa firem a dispečerů.

Dispečer webové aplikace po přihlášení bude moci vytvářet a spravovat pracovníky, kteří náleží do jeho firmy. Jeho hlavní činností bude vytvářet a spravovat úkoly. Tyto úkoly se budou dělit do několika kategorií, které jsou blíže popsány v kapitole 3.4. Jedná se o:

- šablony úkolů,
- přiřazené úkoly,
- hlavní úkoly.

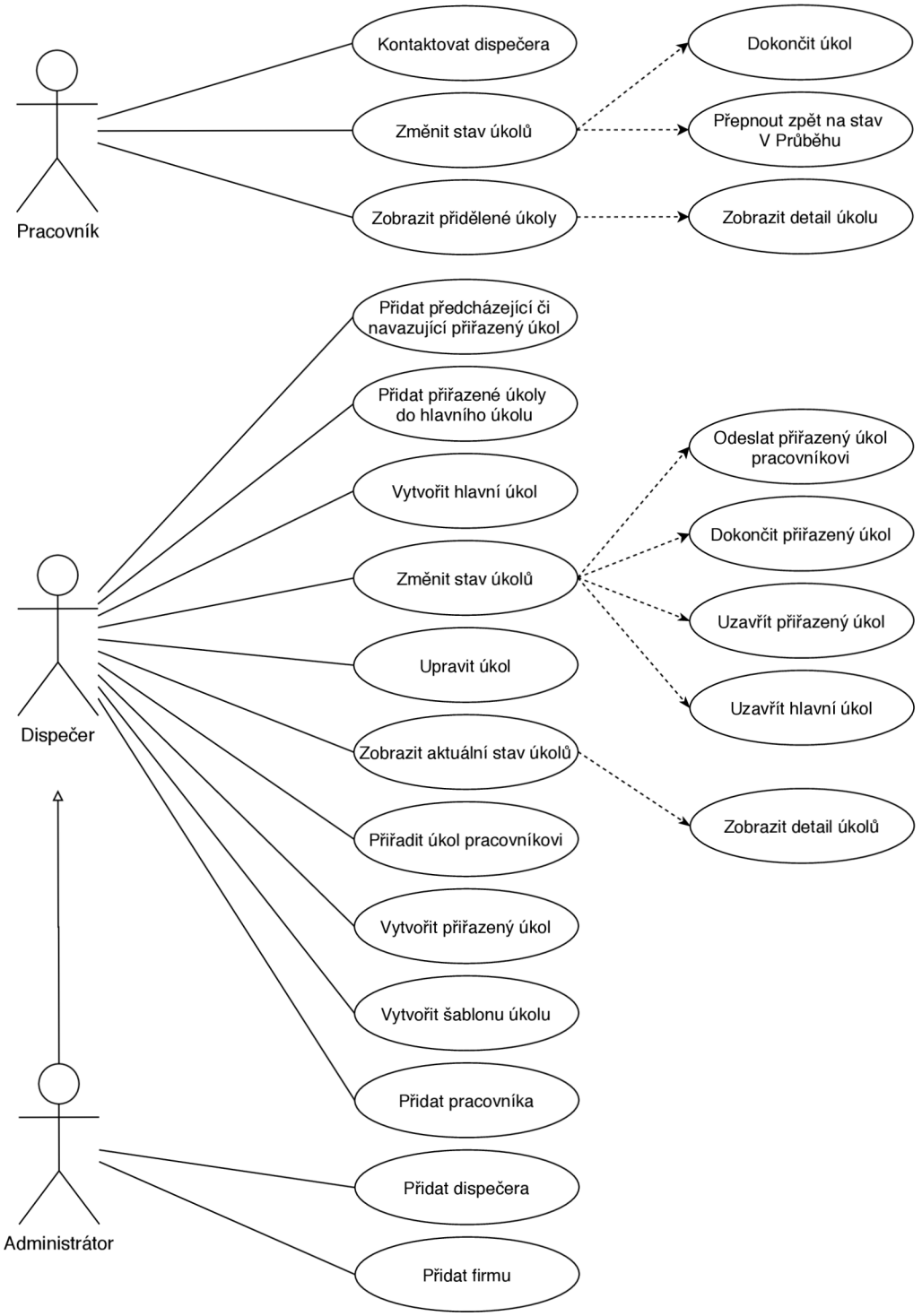
3.3 Návrh databáze

Pro návrh databáze jsem použil ERD[1] na obrázku 3.2. Pro tvorbu tohoto diagramu jsem použil volně dostupný nástroj draw.io[2]. Diagram popisuje jednotlivé entity databázového modelu a vztahy mezi nimi.

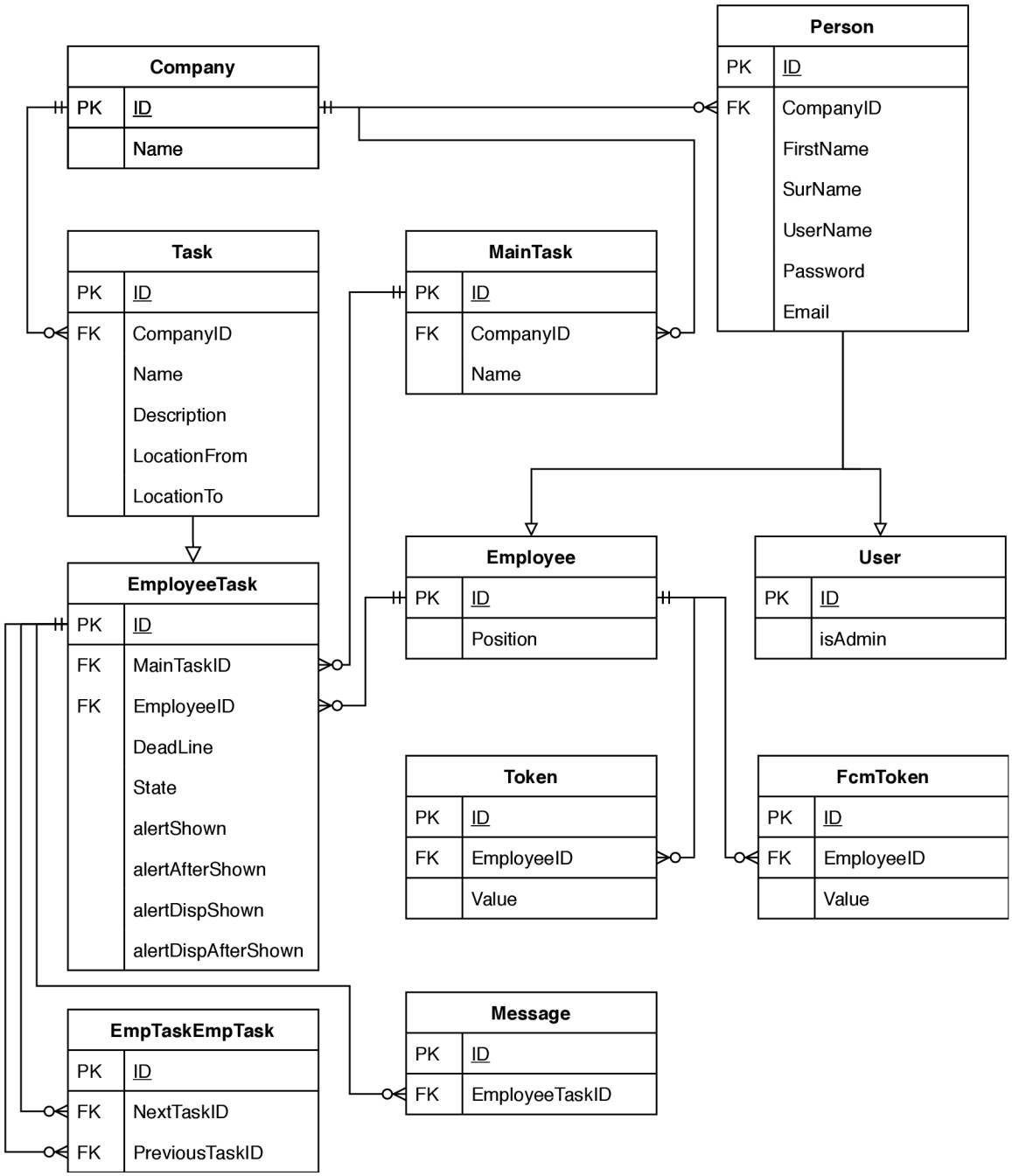
Popis jednotlivých entit ERD z obrázku 3.2.

- **Company** – Hlavní účel této entity bude zapouzdření všech dispečerů, pracovníků a jejich úkolů, jež náleží do jedné firmy. Díky této entitě bude možné, aby aplikaci používalo více firem najednou, přičemž každá firma vidí pouze svá data.
- **Person** – Tato entita bude popisovat obecné informace o uživatelích aplikace. A to jak webové aplikace, tak i mobilní aplikace. Vlastnosti dědí entity Empoloyee a User.

[1] Entity Relationship Diagram (ERD) – Entitně vztahový diagram sloužící pro návrh databáze.
[2] Draw.io – Webový nástroj pro tvorbu diagramů.



Obrázek 3.1: **Diagram případů užití.** Diagram na obrázku ukazuje možné činnosti uživatelů aplikací. Administrátor přidává firmy a dispečery. Dispečer vytváří jednotlivé úkoly a pracovníky a také úkoly přiděluje. Pracovník v mobilním zařízení potvrzuje dokončené úkoly.



Obrázek 3.2: **Návrh databáze.** Na obrázku je ERD model databáze.

- **User** – Entita User bude popisovat účty uživatelů, kteří se budou moci přihlásit do webové aplikace jako dispečeři, a účty administrátorů celé webové aplikace. Právě atribut isAdmin bude sloužit k identifikaci administrátora.
- **Employee** – Tato entita bude popisovat jednotlivé pracovníky používající mobilní aplikaci. Tito pracovníci nebudou mít přístup do webové aplikace.
- **Token** – Entita token bude popisovat tokeny pracovníků, které budou sloužit pro přistupování pracovníků na API webové aplikace.
- **FcmToken** – FcmToken je bude entita popisující tokeny služby Firebase Cloud Messaging (kapitola 2.4), které budou sloužit pro odesílání notifikací z webové aplikace do mobilní aplikace.
- **Task** – Bude entita popisující šablony úkolů. Tyto šablony dispečer vytvoří a použije v případě přiřazování úkolů podobných parametrů.
- **EmployeeTask** – Bude obsahovat úkoly přiřazené pracovníkům. V prvním návrhu ERD sloužila entita EmployeeTask jako vazební mezi entitami Task a Employee. Kvůli nutnosti editace přiřazeného úkolu byla entita změněna na specializaci entity Task.
- **EmpTaskEmpTask** – Vazební tabulka pro vztah následujících a předcházejících entit EmployeeTask.
- **MainTask** – Tato entita bude popisovat sdružení přiřazených úkolů do jednoho nadřazeného úkolu. Entita bude sloužit primárně pro přehlednost při práci dispečera.

3.4 Webová aplikace

Dispečer bude mít k dispozici webovou aplikaci poskytující rozhraní pro správu pracovníků a jejich úkolů. Tato aplikace musí dispečerovi umožňovat efektivně vytvářet a přidělovat úkoly. Dispečer musí mít také neustálý přehled o stavu přidělených úkolů.

3.4.1 Typy úkolů

Aplikace obsahuje několik typů úkolů:

- **Šablony úkolů** – Tyto úkoly budou sloužit pro uložení úkolů, které se často opakují. Dispečer tak opakovaně nebude muset vyplňovat stejné informace. Příkladem úkolu může být odvezení kontejneru písku. Dispečer zvolí příslušnou šablonu, doplní lokaci a vybere pracovníka, kterému tento úkol přidělí.
- **Přidělené úkoly** – Jedná se o konkrétní úkoly přidělené pracovníkům. Mohou vzniknout vytvořením ze šablony, nebo se dají vytvořit přímo pomocí formuláře. Díky různým stavům, které bude moci úkol nabývat, může dispečer tento úkol vytvořit už při přijímání objednávky, kdy nebude vědět, zda se informace či pracovník, kterému je úkol přidělen, do budoucna ještě nezmění. Úkoly budou totiž po vytvoření ve stavu „V přípravě (10)“, který se neodesílá, ani nezobrazuje pracovníkovi v aplikaci. Dispečer si tedy úkol může uložit, jeho informace doplnit později a až je úkol připraven, tak ho odešle pracovníkovi.

- **Hlavní úkoly** – Tyto úkoly slouží pro zvýšení přehlednosti. Jejich účel bude sdružovat přidělené úkoly a zjednodušit tak orientaci dispečera při práci s komplexnějšími úkoly, které budou v aplikaci reálně rozděleny do více podúkolů. Detail hlavních úkolů také bude obsahovat velice zjednodušený graf přehledu návaznosti všech jeho podúkolů. Protože přiřazený úkol (podúkol hlavního úkolu) bude moci obsahovat libovolný počet následovníků i předchůdců, je tento pohled efektivním zjednodušením přehledu vazeb mezi úkoly.

3.4.2 Stavy úkolů

Jako první bylo třeba vyřešit workflow úkolů a s tím související stavy úkolů. Po několika návrzích a následných změnách bylo dosaženo ideální posloupnosti stavů. Tyto stavy jsou:

- „V přípravě (10)“. V tomto stavu se bude nacházet úkol po vytvoření dispečerem. Dispečer v tuto chvíli bude vyplňovat a měnit parametry úkolu. V tomto stavu úkol nebude muset být přiřazený pracovníkovi. V případě, že bude mít úkol přiřazeného pracovníka, tak pracovník úkol ještě neuvidí a ani o něm nebude nedostávat žádnou notifikaci.

- „V pořadí (30)“. Do tohoto stavu se přepne úkol, pokud bude existovat alespoň jeden úkol, který bude danému úkolu přidaný jako předchozí úkol, který bude stavu menším, než „Dokončen (70)“.

- „V řešení (50)“. Tento stav se úkolu nastaví ve dvou případech. První případ bude, že dispečer klikne na tlačítko odeslat, úkol má nastavené potřebné parametry a nemá žádné předcházející nedokončené úkoly. Druhým případem bude úkol, který bude ve stavu „V pořadí (30)“ a všechny jeho předcházející úkoly byly dokončeny, poté se automaticky změní stav úkolu na „V řešení (50)“.

- „Dokončen (70)“. Do tohoto stavu se úkol dostane ze stavu „V řešení (50)“ při kliknutí na tlačítko Dokončit. Stav značí dokončený úkol.

- „Uzavřen (80)“. Tento stav značí, že je úkol dokončen a už na něm nebudou prováděny žádné změny. Alternativně by se dal označit jako archivován. Takový úkol se už nebude nezobrazovat v hlavním přehledu úkolů, ale budou pro ně speciální obrazovky.

Dalším funkcí bylo vytvoření logiky ukládání při přidělování úkolů dle zadaných požadavků. V zadání bylo totiž požadováno, aby si dispečer mohl vytvořit šablony úkolů pro rychlejší přidělování opakovaných úkolů. Aplikace ale musí implementovat možnost následného individuálního upravování přidělených úkolů.

V prvním návrhu řešení jsem tento problém implementoval jako vytvoření šablony úkolu s následným odkazem na příslušnou šablonu v databázi při přidělení úkolu pracovníkovi. Toto řešení bylo úsporné z hlediska prostorové náročnosti databáze, ale představovalo komplikaci při následném upravování přidělených úkolů. Pokud byla upravena šablona či úkol z ní vytvořený, pak byla upravena šablona včetně všech úkolů z ní vycházejících.

Řešením tohoto problému je vytvoření samostatného úkolu (místo vazby na šablonu úkolu) při změně parametrů přiděleného úkolu. Tato změna by však vedla k přílišné složitosti a dvojímu možnému uložení přiřazených úkolů v databázi. Jedna možnost jako odkaz na šablonu a druhá jako samostatná zduplikovaná data bez odkazu.

Jako konečné řešení tohoto problému bylo tedy zvoleno automatické vytvoření duplicitního úkolu v databázi při přiřazování úkolu uživateli z šablony. Toto řešení vyžaduje

duplicitní uložení některých dat v databázi, avšak i s touto nevýhodou má značné výhody v oblasti aplikační práce s úkoly.

Díky tomuto způsobu uložení přiřazených úkolů v databázi bude možné také jednoduše vytvářet přiřazené úkoly bez nutnosti existence šablony. Tuto možnost využije dispečer v případech, kdy bude vytvářet úkoly s ojedinělými parametry.

3.5 Mobilní aplikace

Největší důraz bude kladen na jednoduchost a přehlednost mobilní aplikace. V úvahu je třeba vzít fakt, že možné využití aplikace je i při řízení auta či nákladního vozidla. Proto aplikace na hlavní stránce, kterou bude přehled úkolů, bude zobrazovat minimalistické informace o úkolech a poskytovat jednoduché rozhraní pro přepnutí stavů těchto úkolů.

Uživatel bude muset mít ale také možnost zobrazení detailu úkolu a v něm mít přehled o všech informacích patřících k úkolu.

Aplikace bude muset obsahovat notifikace o nově příchozím úkolu, aby uživatel nemusel neustále kontrolovat, zda mu dispečer nějaký úkol nepřidělil. Tyto notifikace by měly fungovat jak při minimalizované či ukončené aplikaci, tak i při spuštěné a zobrazené aplikaci.

Součástí bude také možnost rychlého kontaktování dispečera bez nutnosti telefonického hovoru. Využití této funkce nastane například při poruše, kdy pracovník musí co nejrychleji vyřešit problém, ale zároveň potřebuje upozornit dispečera na komplikace, aby bylo možné včas přeorganizovat úkoly. Upozornění bude poskytovat možnost zadání krátké zprávy pro lepší identifikaci problému.

Do aplikace se budou odesílat úkoly ve stavu „V pořadí (30)“, „V řešení (50)“ a „Dokončen (70)“. Dokončené úkoly se v aplikaci budou zobrazovat pouze do doby, než vyprší termín jejich dokončení. Je to proto, aby pracovník mohl po označení úkolu jako dokončený jeho stav vrátit a zároveň aby se v aplikaci nepletly dokončené úkoly, které jsou po termínu a dispečer je ještě nepřepl do stavu „Uzavřen (80)“.

3.6 Uživatelské rozhraní

Při návrhu uživatelského rozhraní byly využity znalosti respektující pravidla uvedené v kapitole 2.3. Také byl brán ohled na uživatelské rozhraní implementované v podobných aplikacích popsaných v kapitole 2.7, například aplikace Todoist, představené v kapitole 2.7.1.

3.6.1 Koncepce uživatelského rozhraní

Uživatelské rozhraní obou aplikací se bude snažit dodržovat zaběhlé zvyklosti. Důvodem je zkrácení procesu porozumění aplikaci, protože uživatelé budou intuitivně vědět kam kliknout. Při dnešním rozmachu informačních technologií je těžké vymyslet efektivnější a uživatelsky více přijatelné řešení, než existuje doposud. I v případě, že by bylo navrhované řešení lepší než nějaké existující, tak to nutně nemusí znamenat ulehčení práce a zvýšení efektivity, protože jsou uživatelé zvyklí na něco jiného.

Zároveň je kladen důraz na jednoduchost a čistotu uživatelského rozhraní. Je zde snaha o minimální potřebné rozhraní bez zbytečných tlačítek a informací.

Webová aplikace

Jednou z nejdůležitějších částí aplikace je přehled přiřazených úkolů. Tato část bude zobrazovat tabulku se seznamem přiřazených úkolů, které budou ve stavu „V pořadí (30)“ a „V řešení (50)“. Úkoly na obrazovce budou seskupovány dle pracovníků. Dispečer bude mít tedy přehled o aktuální vytíženosti podřízených pracovníků a stavu jejich úkolů. Obrazovka u každého úkolu bude obsahovat stejné tlačítko, jaké se zobrazí pro změnu stavu na obrazovce detailu úkolu. Dispečer tedy nebude muset přecházet na detail úkolů, aby změnil jejich stav. V tabulce bude zobrazeno číslo úkolu, název, termín, hlavní úkol a stav.

Další důležitou částí aplikace budou seznamy úkolů a jejich detaily. Tyto úkoly budou různých typů a každý typ úkolů bude mít svou vlastní obrazovku se seznamem a vlastní obrazovku detailu. Typy úkolů jsou popsány v kapitole 3.4.1.

- **Šablony úkolů** – tento typ úkolů nebude neobsahovat stavy. Seznam těchto úkolů bude obsahovat pouze název úkolu, popis, počáteční lokaci a cílovou lokaci. K těmto atributům úkolu bude na každém řádku přidáno tlačítko pro zobrazení detailu úkolu.

  Obrazovka detailu úkolu bude obsahovat tlačítka na upravení a smazání úkolu. Dále zde bude tlačítko pro přiřazení šablony úkolu pracovníkovi a tím vytvoření přiřazeného úkolu s parametry šablony.

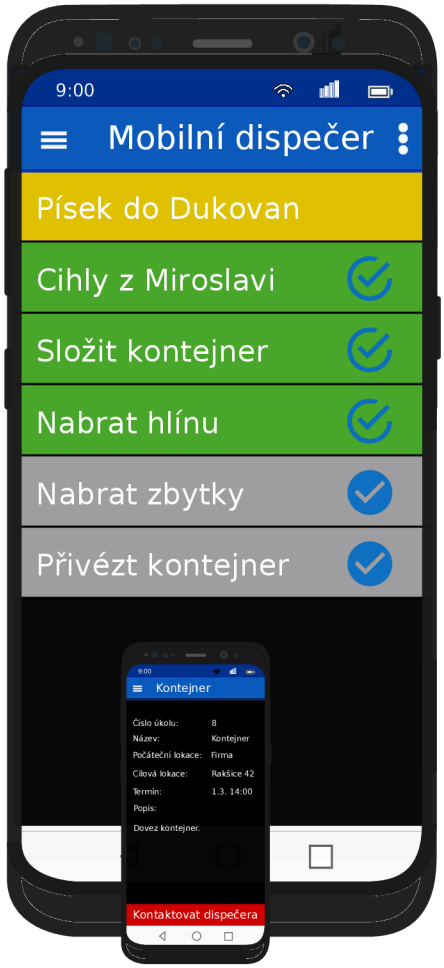
- **Přiřazené úkoly** – seznam těchto úkolů bude také rozdělen na dva seznamy. Seznam uzavřených přiřazených úkolů a hlavní seznam. Hlavní seznam bude obsahovat atributy: číslo, název, termín, pracovník, hlavní úkol a stav. Kromě atributů bude obsahovat také tlačítka pro možnost přepnutí stavu úkolu bez nutnosti otevírání detailu úkolu. Součástí bude také tlačítko pro zobrazení jeho detailu. Řádky tabulky budou barevně podbarveny dle stavu úkolu pro lepší vizuální přehled.

  Detail přiřazeného úkolu bude obsahovat seznam všech atributů, předchozích a následujících úkolů a název hlavního úkolu. Detail také bude obsahovat větší množství tlačítek:

  - Upravit úkol.
  - Přidat předchozí úkol.
  - Přidat následující úkol.
  - Přidat do hlavního úkolu.
  - Přiřadit pracovníkovi
  - Tlačítka pro změnu stavu úkolu.

- **Hlavní úkoly** – pro tento typ úkolu budou vytvořeny dva samostatné seznamy. Jeden seznam bude sloužit pro přehled úkolů, se kterými se bude pracovat a druhý seznam bude obsahovat uzavřené úkoly. Je to z důvodu, aby se v seznamu hlavních úkolů nezobrazovaly zbytečně úkoly, které už budou uzavřené a dispečer s nimi už pravděpodobně nikdy pracovat nebude.

  Detail hlavního úkolu bude obsahovat tlačítka pro upravení úkolu, přidání dílčího úkolu a za splnění určitých podmínek také tlačítko pro uzavření úkolu. Součástí detailu bude také seznam přiřazených dílčích úkolů a prototyp mapy návaznosti dílčích úkolů.



Obrázek 3.3: **Návrh mobilní aplikace.** Na obrázku vlevo je znázorněno minimalistické rozhraní při zobrazení seznamu úkolů. U jednotlivých úkolů budou tlačítka pro změnu stavu. Na obrázku vpravo je znázorněn detail úkolu a ve spodní části tohoto obrázku se nachází tlačítko pro kontaktování dispečera.

Mobilní aplikace

Hlavním cílem mobilní aplikace má být rychlý a jednoduchý přehled přiřazených úkolů a jejich potvrzování a tomuto faktu bude přizpůsobeno uživatelské rozhraní. Aplikace se bude skládat ze dvou obrazovek.

První obrazovka bude přehled přiřazených úkolů (obrázek 3.3 vlevo). Na této obrazovce se bude nacházet seznam úkolů a jejich základní informace. V případě nutnosti získání podrobnějších informací o úkolu může uživatel kliknutím na příslušný úkol zobrazit jeho detail. V pravé části každé položky úkolu v seznamu se bude nacházet tlačítko pro změnu jeho stavu. Po kliknutí na tlačítko pro změnu stavu bude muset uživatel volbu potvrdit. Je to bezpečnostní opatření proti nechtěné změně stavu úkolu. Pracovník bude mít také na hlavní stránce seznamu úkolů k dispozici filtr, který poskytuje možnost zobrazit pouze úkoly ve stavu „V řešení (50)“ (výchozí možnost) či zobrazit všechny dostupné úkoly odeslané do mobilní aplikace. Tato možnost bude zde opět pro zjednodušení a zpřehlednění aplikace.

Obrazovka detail úkolu pracovníkovi bude zobrazovat veškeré dostupné informace o úkolu (obrázek 3.3 vpravo). Ve spodní části obrazovky bude tlačítko pro rychlé kontaktování dispečera v případě nouze.

3.6.2 Vývoj uživatelského rozhraní

Uživatelské rozhraní mobilní aplikace se oproti původnímu návrhu (v kapitole 3.6) změnilo pouze lehce. Vzhled přehledu všech úkolů se zjemnil a zmodernizoval a na detail úkolu přibylo tlačítko pro změnu stavu úkolu.

Webová aplikace byla oproti první verzi v mnohém změněna. Hlavní změny byly provedeny v přidávání součástí nově požadovaných funkcí aplikace, které nebyly při jejím prvním zadávání plánovány a poté se zjistilo, že jsou v aplikaci potřeba. Změny dále tvořila reorganizace komponent aplikace. V rámci testování se ukázalo, že některé kroky v aplikaci jsou zbytečně složité a mohou být zjednodušeny přesunutím či duplikací tlačítka na jinou obrazovku. Po celou dobu vývoje aplikace byla soustředěna pozornost mnohem více na funkcionalitu, než na vzhled. V poslední fázi vývoje tedy aplikace podstoupila zásadní vylepšení vzhledu.

3.7 Přednosti mého řešení

Mé řešení představuje obrovské ulehčení práce dispečera i pracovníků oproti současným možnostem. Aplikace poskytují nástroje pro rychlé přidělování úkolů, okamžitý přehled o aktuálních stavech úkolů a to jako v mobilní, tak i ve webové aplikaci a další funkce zefektivňující tento proces práce.

Výhodou aplikace je například upozornění na úkol blížící se požadovanému termínu dokončení a na úkol, kterému termín vypršel. Tato funkcionalita funguje jak v dispečerské webové aplikaci, tak i v mobilní aplikaci pracovníků. Součástí mobilní aplikace je také možnost odeslat dispečerovi krátkou zprávu, která se váže k jednomu konkrétnímu úkolu. Tuto možnost pracovník využije v případě, že nastal problém, například prasknutí pneumatiky, a potřebuje problém rychle řešit a zároveň informovat dispečera.

Úkoly mohou mít mezi sebou vazby ve smyslu předcházejícího a následujícího úkolu. Počet těchto vazeb u každého úkolu není omezen. Úkol tedy může mít například 3 následující a 4 předcházející úkoly. Dispečer má také možnost sdružování jednotlivých úkolů do skupin. Pokud se úkoly nachází ve skupině, je k dispozici graf návaznosti úkolů pro zlepšení přehledu složitých návazností.

V celém průběhu vývoje aplikace byl kladen důraz na její univerzálnost. Vznikla tak aplikace poskytující vhodné nástroje k řešení problému přidělování úkolů pracovníkům dispečerem, ale aplikaci je možné použít také v mnoha jiných dalších odvětvích. V rámci testování byla univerzálnost aplikace otestována. Testování je popsáno v kapitole 4.11.

Kapitola 4

Implementace a vyhodnocení

Webová aplikace je implementována v jazyce PHP, konkrétně ve frameworku Symfony. Je zde použito CSS a Bootstrap pro stylování stránky a JavaScript pro doplnění funcionalit. Mobilní aplikace je implementována ve frameworku Flutter, který je založen na jazyce Dart.

4.1 Základní koncepce díla

Dispečerská webová aplikace slouží k vytváření pracovníků a přidělování úkolů daným pracovníkům. Aplikace má přímý přístup k databázi, kam se ukládají veškerá data. Mobilní aplikace veškerá data získává pomocí přístupu na REST API webové aplikace. V případě přihlášení do mobilní aplikace se do webové aplikace odešlou přihlašovací údaje. V případě úspěšného přihlášení vygeneruje webová aplikace token, pomocí kterého se následně mobilní aplikace identifikuje při přístupu na API webové aplikace. Jedná se například o načtení a zobrazení úkolů příslušného pracovníka, změny stavů úkolů pomocí mobilní aplikace, nebo kontaktování dispečera.

4.2 Databáze

Struktura databáze je definována pomocí **entit**. Entita je speciální třída, která zpravidla obsahuje definici pro jednu tabulku v databázi. Pomocí speciální syntaxe jsou zde definovány jednotlivé atributy (sloupce) tabulky. Těmto atributům je zde určen typ. Dále se u atributů dá nastavit například, zda hodnota daného atributu může být NULL, výchozí hodnota, unikátnost v databázi, nebo délka řetězce v databázi. Symfony používá implicitně lazy loading[1] a tato funkcionalita se zde dá také zakázat. Doctrine, které používá Symfony ke správě databáze, obsahuje příkaz `make:entity`. Tento příkaz značně ulehčuje vytváření entit v Symfony. Obsahuje průvodce, ve kterém se postupně zadává název entity, název atributu, datový typ atributu a případné další dodatečné konfigurace dle datového typu. Tento příkaz zároveň automaticky vytvoří `get` a `set` metody pro práci s těmito atributy.

Pro vývoj a testování byla použita lokální databáze. Pro použití databáze je vyžadována konfigurace. Tuto konfiguraci jsem nastavil v souboru `.env` připsáním jednoho řádku:
`DATABASE\_URL=mysql://root:@127.0.0.1:3306/bakalarka`.
K vytvoření lokálního databázového serveru jsem použil program XAMPP. Tento program

[1] **Lazy loading** – optimalizační technika načítání dat z databáze. Více zde: doctrine-project.org/projects/doctrine-orm/en/2.7/tutorials/extra-lazy-associations.html

poskytuje jak Apache server pro vývoj PHP aplikací v lokálním prostředí, tak i již zmíněný MySQL server pro databázi.

V produkčním nasazení aplikace běží na službě Heroku[2], která obsahuje doplněk **Jawsdb**. Jedná se o samostatný MySQL databázový server. K vytvoření této databáze je třeba ve složce, která je nastavena jako kořen aplikace Heroku, zadat příkaz:
`heroku addons:create jawsdb`.
Poté je příkazem:
`heroku config:get JAWSDB\_URL`,
vypsána adresa právě vytvořeného serveru. Posledním krokem je nastavení proměnné DATABASE_URL na serveru pomocí přikazu:
`heroku config:set DATABASE\_URL={adresa vypsána předchozím příkazem}`.
Tímto je nastaven databázový server pro aplikaci běžící na Heroku.

Po nastavení příslušných serverů je třeba vytvořit potřebnou databázovou strukturu. Pro vytvoření databáze slouží příkaz:
`php bin/console doctrine:database:create`.
O strukturu databáze se stará Doctrine pomocí databázových migrací. Ty jsou buď již vygenerované nebo stačí zadat příkaz:
`php bin/console doctrine:migrations:diff`
pro vytvoření databázové migrace a poté:
`php bin/console doctrine:migrations:migrate`
pro uskutečnění migrace.

Struktura databáze byla implementována dle popisu v kapitole 3.3.

4.3 Vzhled

4.3.1 Použití twig ve webové aplikaci

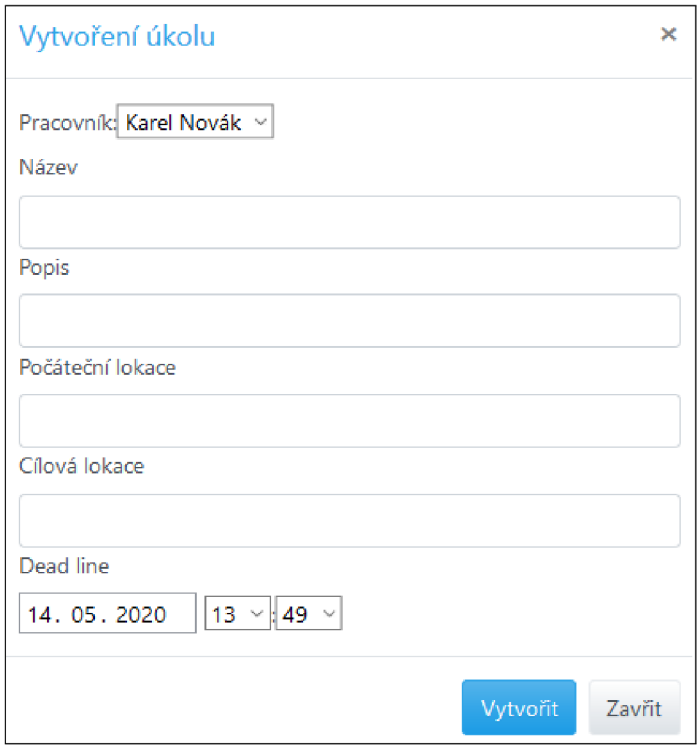
Symfony používá šablonový systém Twig[3]. Použití těchto šablon umožňuje oddělit aplikační a prezentační vrstvu aplikace. Vývoj aplikace pomocí těchto šablon je velice pohodlný. Šablony jsou implementovány v souborech s příponou**html.twig**. Základem každé stránky je soubor `base.html.twig`, který definuje základní prvky výsledného HTML dokumentu. Soubor `base.html.twig` obsahuje část `{% block body %}{% endblock %}`, která se nahradí obsahem příslušné šablony. V šabloně je nutné zadat `{% extends 'base.html.twig' %}`. Poté je třeba celý kód vložit do `{% block body %}{% endblock %}`. Tímto se provede vložení upřesňující šablony do základní šablony. Titulek stránky je implementovaný stejně. Pro ten je zde `{% block title%}`.

Požadované šablony se poté volají z kontrolerů, kde se do šablon pošlou i jejich proměnné.

4.3.2 Webová aplikace

Pro určení téma vzhledu webu je použit **Bootswatch**[4]. Tento doplněk velice usnadňuje vývoj nastavením jednotného vzhledu všem důležitým prvkům aplikace. Nemusí se tedy

[2] **Heroku** – heroku.com
[3] **Twig** – twig.symfony.com
[4] **Bootswatch** – volně dostupná témata pro Bootstrap



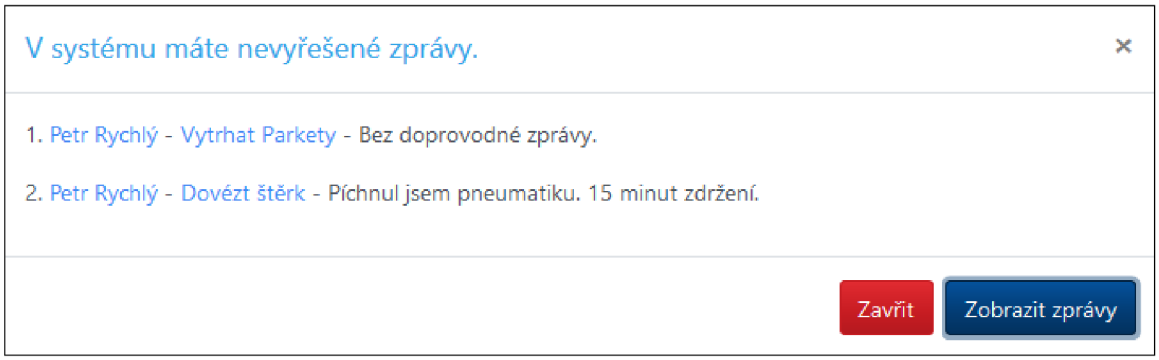
Obrázek 4.1: **Modální okno** – Na obrázku je zobrazeno modální okno pro vytvoření přiřazeného úkolu. Uživatel zde vyplní požadované parametry a klikne na tlačítko „Vytvořit“. Pokud uživatel nevyplní povinná pole, případně nastanou konflikty v polích, u kterých je požadována unikátnost, pak se u jednotlivých polí zobrazí příslušné upozornění.

pro každý prvek rozhraní psát vlastní definice vzhledu. Nebo také je díky tomu možné snadno a rychle změnit vzhled celého webu. Možnost libovolné změny vzhledu není implementována, ale je možné na základě uživatelského požadavku díky tomuto doplňku vzhled jednoduše změnit. Pro další stylování a pokročilejší vzhledové funkcionality je použit **Bootstrap** (kapitola 2.5.3). Tento doplněk zabezpečuje například rozbalovací menu v horní části aplikace, vzhled tlačítek nebo modální okna formulářů. Pro některé prvky stránky bylo třeba použít specifického vzhledu, a proto jsou Bootswatch a Bootstrap doplněny o **vlastní CSS**[5]. Pro některé části aplikace je použit vlastní Javascriptový kód.

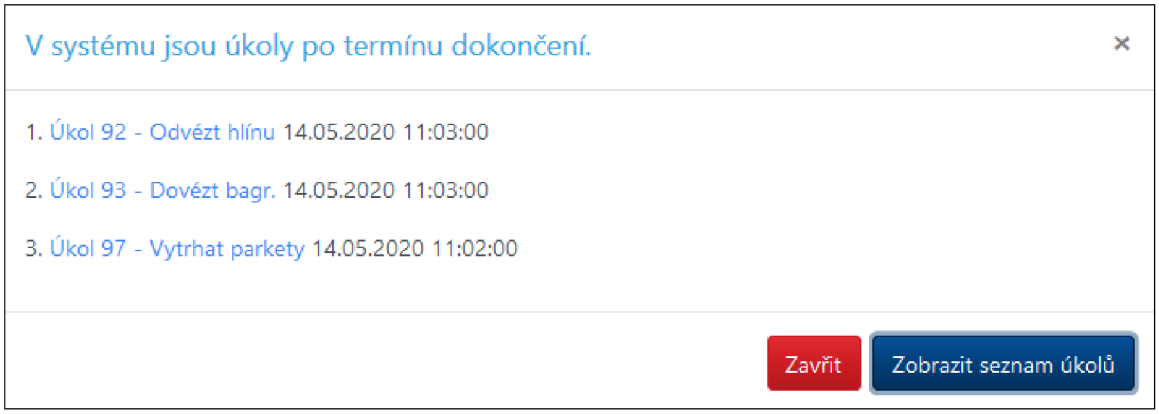
Formuláře v aplikaci jsou implementovány jako **modální okna**. Tento aspekt není klíčový pro funkčnost webu, ale vytváří mnohem lepší dojem při práci s aplikací. Při zobrazování formuláře není nutné přesměrovávat uživatele na další stránku. Místo toho se přes plochu aktuální stránky vykreslí modální okno.Příklad modálního okna se nachází na obrázku 4.1.

Pro upozornění na zprávy a úkoly blížící se termínu dokončení či úkoly po stanoveném termínu byla použita **speciální dialogová okna**. K tvorbě těchto oken byl použit **BootBox**[6], díky kterému je možné v modálním okně mít vlastní tlačítka. Implicitní upozorňovací dialog obsahuje pouze tlačítka „Ok“ a „Zrušit“. Upravit původní dialog je zakázáno z důvodu bezpečnosti. V aplikaci je implementována funkcionalita, která uživateli umožní při

[5]**Cascading Style Sheets (Kaskádové styly)** – jazyk pro definici vzhledu elementů HTML kódu.
[6]**Bootbox** – Javascriptová knihovna pro tvorbu dialogových oken.



Obrázek 4.2: **Vlastní dialogové okno 1** – Na obrázku je vlastní dialogové okno s upravenými tlačítky sloužící pro notifikaci o příchozí zprávě. V horní části je obecná informace o upozornění. Pod ní se nachází seznam zpráv. Ve spodní části jsou tlačítka pro zavření nebo přejití na seznam zpráv.



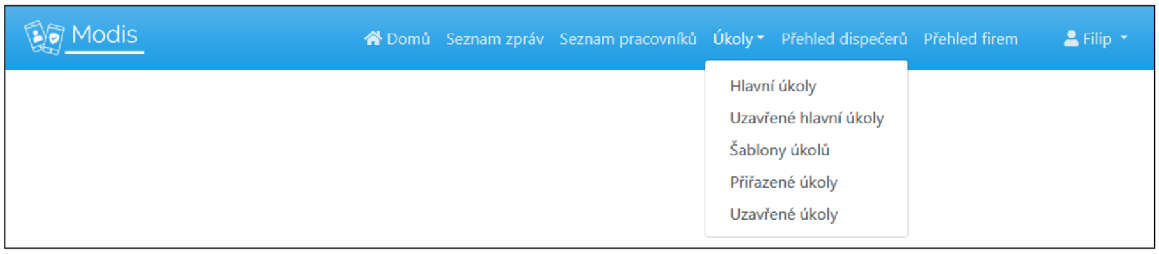
Obrázek 4.3: **Vlastní dialogové okno 2** – Na obrázku je vlastní dialogové okno s upravenými tlačítky sloužící pro notifikaci o úkolech po termínu dokončení. V horní části je obecná informace o upozornění. Pod ní se nachází seznam úkolů po termínu. Ve spodní části jsou tlačítka pro zavření nebo přejití na seznam úkolů.

zobrazení upozorňovacího okna kliknout na tlačítko a přejít na příslušnou stránku. Z tohoto důvodu bylo nutno vytvořit vlastní dialogové okno (obrázek 4.2 a 4.3).

Webová aplikace obsahuje velké množství stránek podobné funkcionality. V této kapitole jsou podrobněji popsány a vyobrazeny pouze zajímavější obrazovky aplikace.

Menu

Menu je vytvořené pomocí Bootstrap, pomocí kterého je možné jednoduše implementovat rozbalovací položky, Ten je doplněn o vlastní CSS. Na obrázku 4.4 je zobrazeno menu, které vidí administrátor aplikace. Toto menu obsahuje i položky „Přehled dispečerů“ a „Přehled firem“, které jsou pro dispečera skryté. V položce „Úkoly“ jsou seskupené všechny položky menu týkající se úkolů. Dále menu obsahuje položky „Seznam pracovníků“ a „Seznam zpráv“. Tyto položky odkazují na příslušné seznamy. Poslední položka menu je „Domů“. Pomocí tohoto tlačítka se dostane uživatel na domovskou obrazovku, kde má přehled aktivních přiřazených úkolů.



Obrázek 4.4: **Menu webové aplikace** – Na obrázku je zobrazeno menu. V levé části je logo a v pravé části je jméno přihlášeného uživatele, kde je po rozkliknutí možnost zobrazení profilu a odhlášení. Uprostřed jsou hlavní položky menu.

Tlačítka

Funkcionalitu tlačítek zařizuje Javascript. Ten tlačítko identifikuje pomocí ID a třídy. ID může obsahovat přímo tlačítko nebo element, do kterého je tlačítko zanořeno (případ více tlačítek vedle sebe). Jednotlivé tlačítka se poté od sebe rozlišují pomocí tříd. Javascript očekává událost kliknutí na tlačítko. Pokud se tak stane, provede nějakou akci. Z pravidla je to fetch na určitou adresu webové aplikace, která provede požadovanou operaci. Pro provedení operace je uživatel buď přesměrován, nebo se zobrazená stránka obnoví.

Řazení v tabulkách

Pro zjednodušení přehledu všech tabulek je v aplikaci implementována Javascriptová funkce, která po kliknutí na libovolný sloupec seřadí řádky tabulky dle tohoto sloupce. Po opětovném kliknutí na stejný sloupec se otočí pořadí řazení. Tato funkce je převzatá ze článku dostupného na stránce GeeksForGeeks[7].

Domovská obrazovka

Domovská obrazovka (obrázek 4.5) obsahuje přehled uživatelů a jejich úkolů ve stavech „V pořadí (30)“ a „V řešení (50)“. Uživateli je umožněn rychlý přístup na detail uživatele či detail úkolu kliknutím na název uživatele nebo číslo a název úkolu.

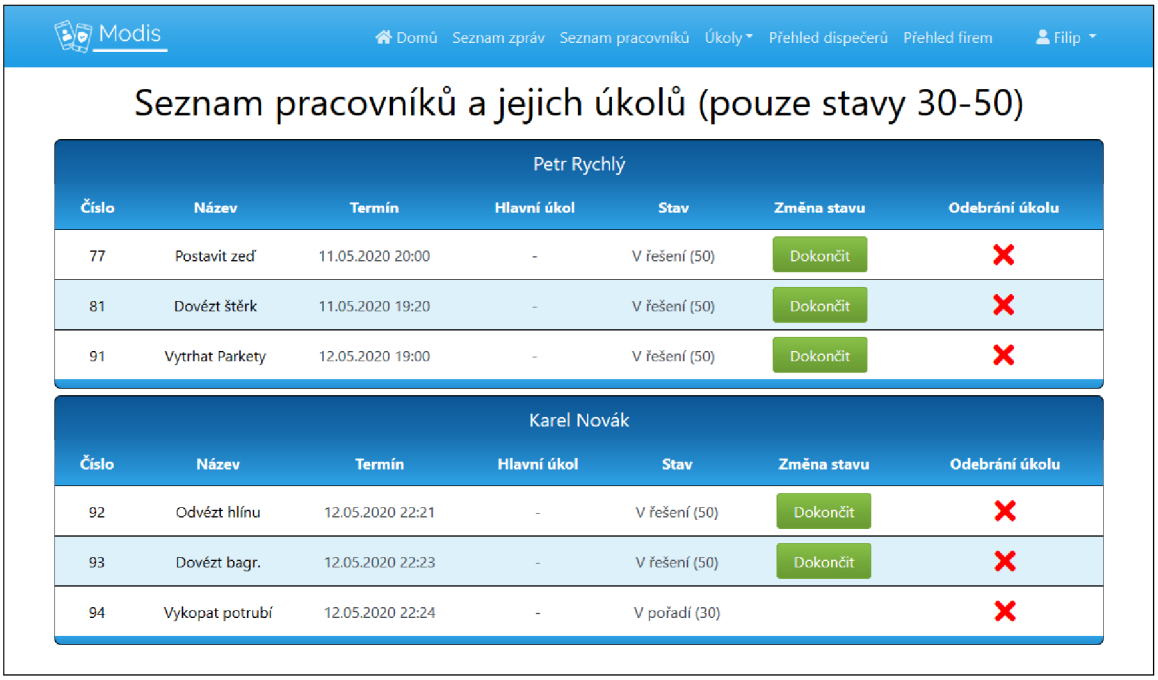
Seznam přiřazených úkolů

Tato obrazovka (obrázek 4.6) slouží jako přehled všech přiřazených úkolů kromě úkolů ve stavu „Uzavřen (80)“. Dispečer pomocí této obrazovky může přidat přiřazený úkol, měnit stavy úkolů či je smazat. Pokud je třeba, může pomocí kliknutí na jméno příslušného pracovníka přejít na jeho detail.

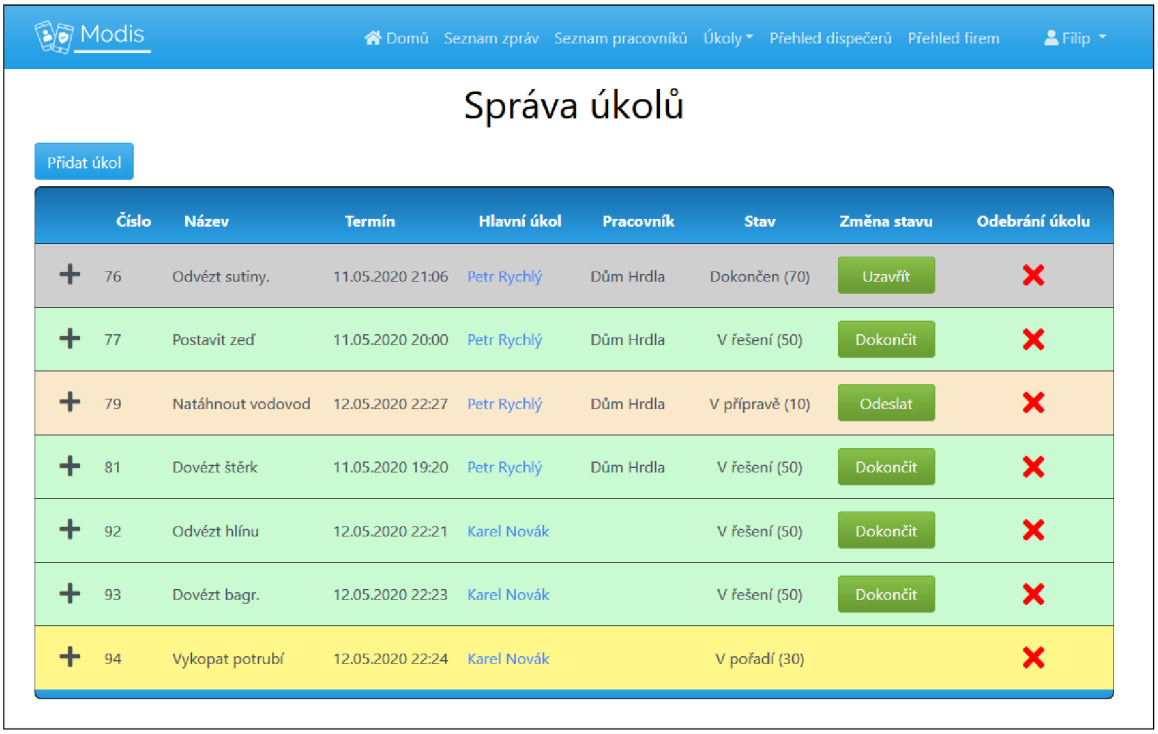
Seznam nevyřízených zpráv

Tato obrazovka (obrázek 4.7) slouží jako přehled všech nevyřízených zpráv. Zprávy se dělí na vyřízené, které se v systému už nezobrazují a nevyřízené, které vidí dispečer v přehledu.

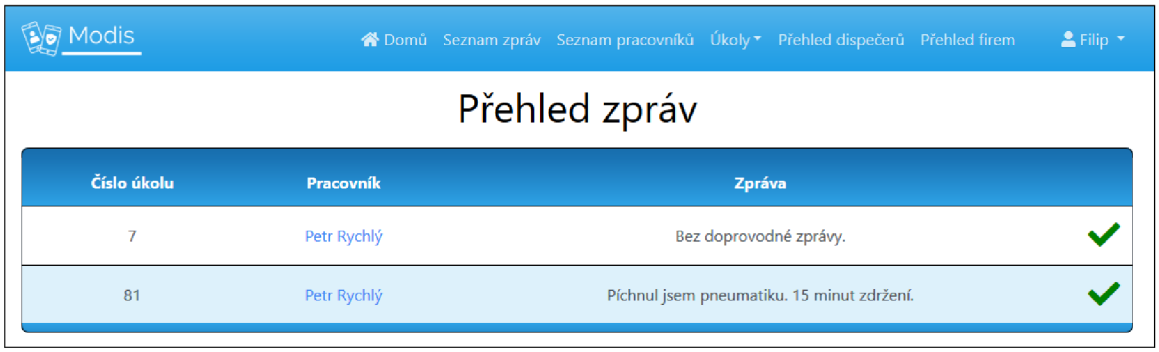
[7] https://www.geeksforgeeks.org/how-to-sort-rows-in-a-table-using-javascript/



Obrázek 4.5: **Domovská obrazovka** – zde se nachází přehled přiřazených úkolů seskupených dle pracovníků. Uživatel má u příslušných úkolů dostupná tlačítka pro změnu stavu úkolů a pro odebrání úkolu.



Obrázek 4.6: **Seznam přiřazených úkolů** – zde se nachází přehled všech přiřazených úkolů. Jednotlivé úkoly jsou od sebe barevně odlišeny dle stavů. Uživatel má k dispozici tlačítka na zobrazení detailu úkolu, změnu stavu úkolu a odebrání úkolu.



Obrázek 4.7: **Seznam zpráv** – zde se nachází přehled nevyřízených zpráv. Dispečer v tabulce vidí číslo úkolu, ke kterému je zpráva přiřazena, pracovníka, který ji odeslal a text zprávy. V pravé části tabulky se poté nachází tlačítko pro označení zprávy jako vyřízené.

Seznam šablon úkolů, firem, pracovníků a dispečerů

Tyto obrazovky obsahují jednoduché tabulkové přehledy se záznamy a jejich atributy. Na každé stránce je pro přidání nového záznamu a u každého záznamu je tlačítko pro zobrazení jeho detailu.

Detail přiřazeného úkolu

Na této obrazovce (obrázek 4.8) je zobrazen detail přiřazeného úkolů a příslušné rozhraní umožňující práci s daným úkolem. Uživatel zde může měnit stav úkolu, přidávat předchozí či následující úkoly a přidat přiřazený úkol do hlavního úkolu.

Detail pracovníka

Obrazovka detail uživatele (obrázek 4.9) slouží k přehledu atributů uživatele, přiřazování úkolů pracovníkovi a jednoduchou správu jeho úkolů. Dispečer se z této obrazovky pomocí tlačítek v seznamu přiřazených úkolů může jednoduše dostat na detail příslušného úkolu.

Detail hlavního úkolu

Obrazovka detailu hlavního úkolu (obrázek 4.10) poskytuje rozhraní pro práci s hlavním úkolem. Největší využití zde představuje prototyp mapy dílčích úkolů. Tyto úkoly mohou mít složité návaznosti a mapa tyto vazby zpřehledňuje pomocí vizuálního zobrazení.

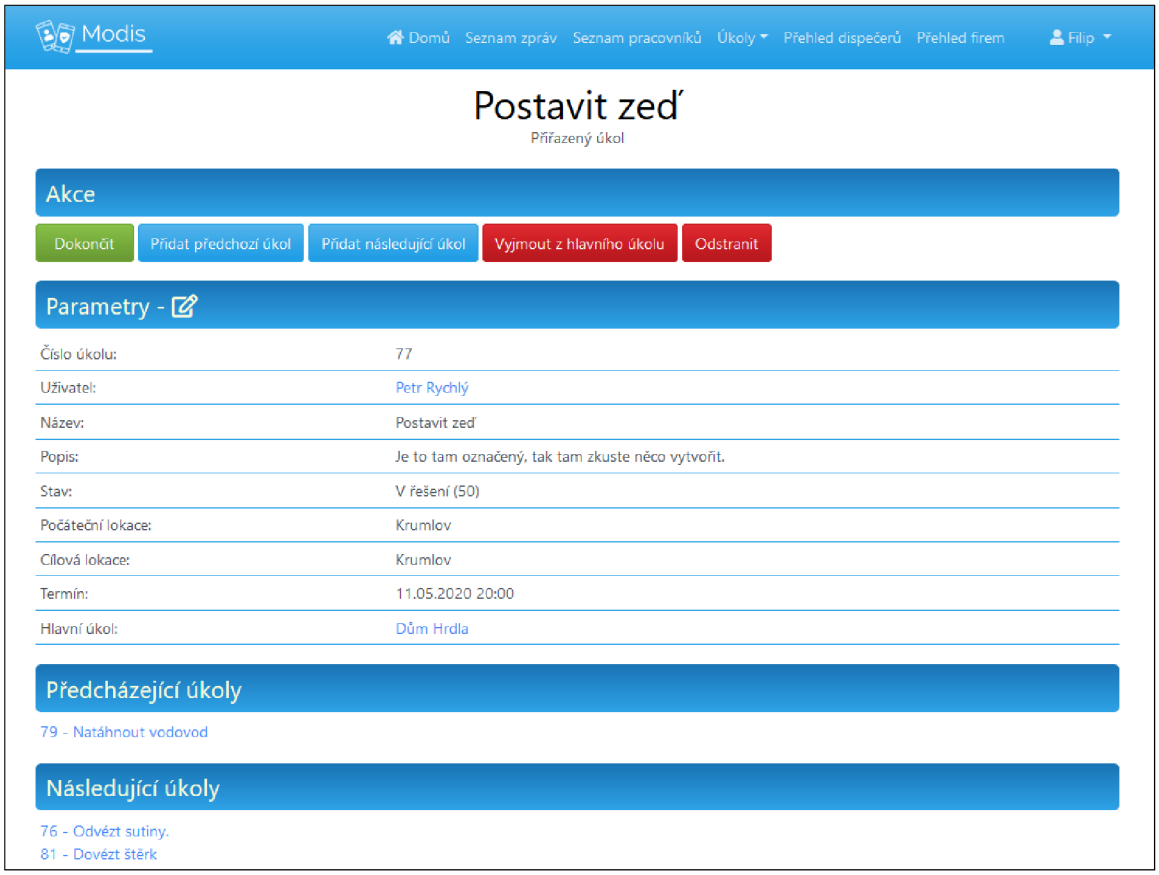
4.3.3 Mobilní aplikace

Pro vzhled mobilní aplikace byl použit **Material Design**[8], který je součástí frameworku Flutter. Použitím tohoto jazyka bylo dosaženo obvyklého vzhledu mobilní aplikace, na který jsou uživatelé zvyklí.

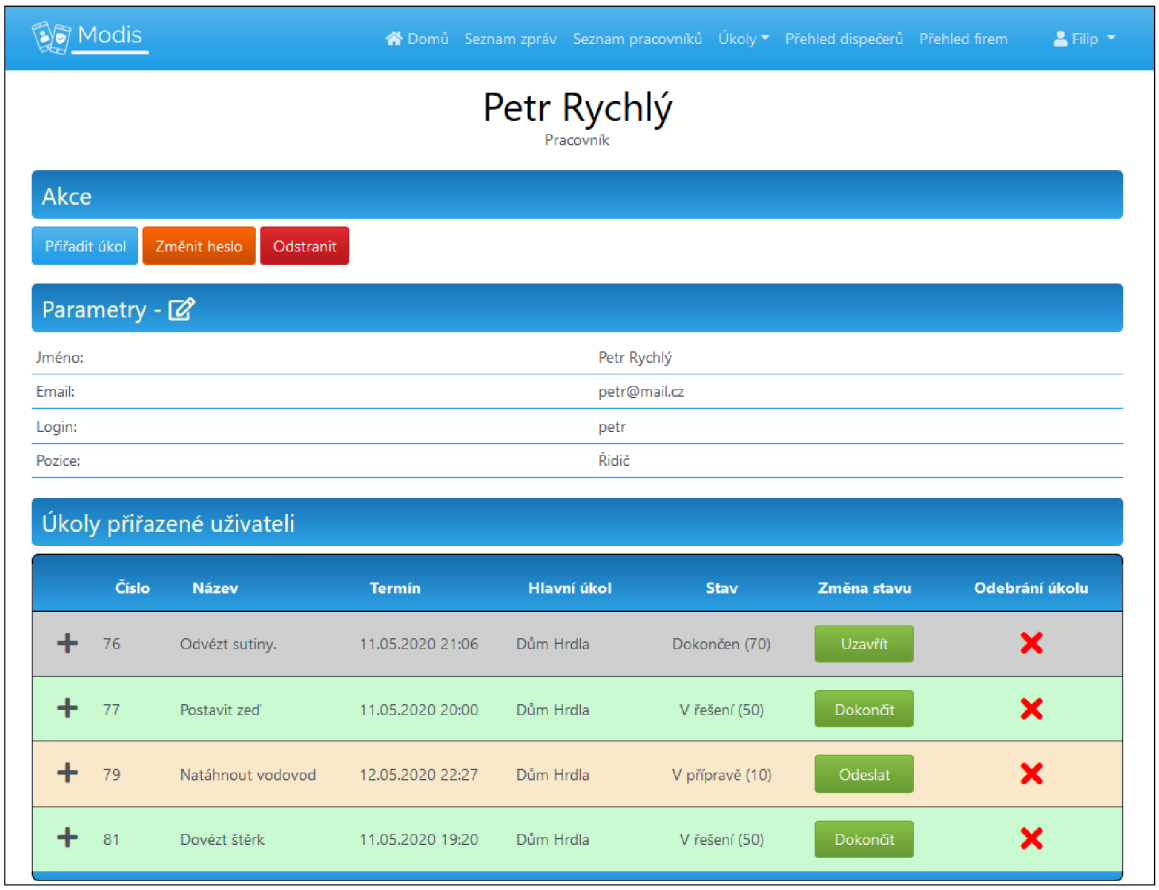
Seznam úkolů

Pro hlavní obrazovku seznamu úkolů je použita komponenta `SingleChildScrollView`, která umožňuje zobrazení delšího seznamu položek, než je výška obrazovky mobilního tele-

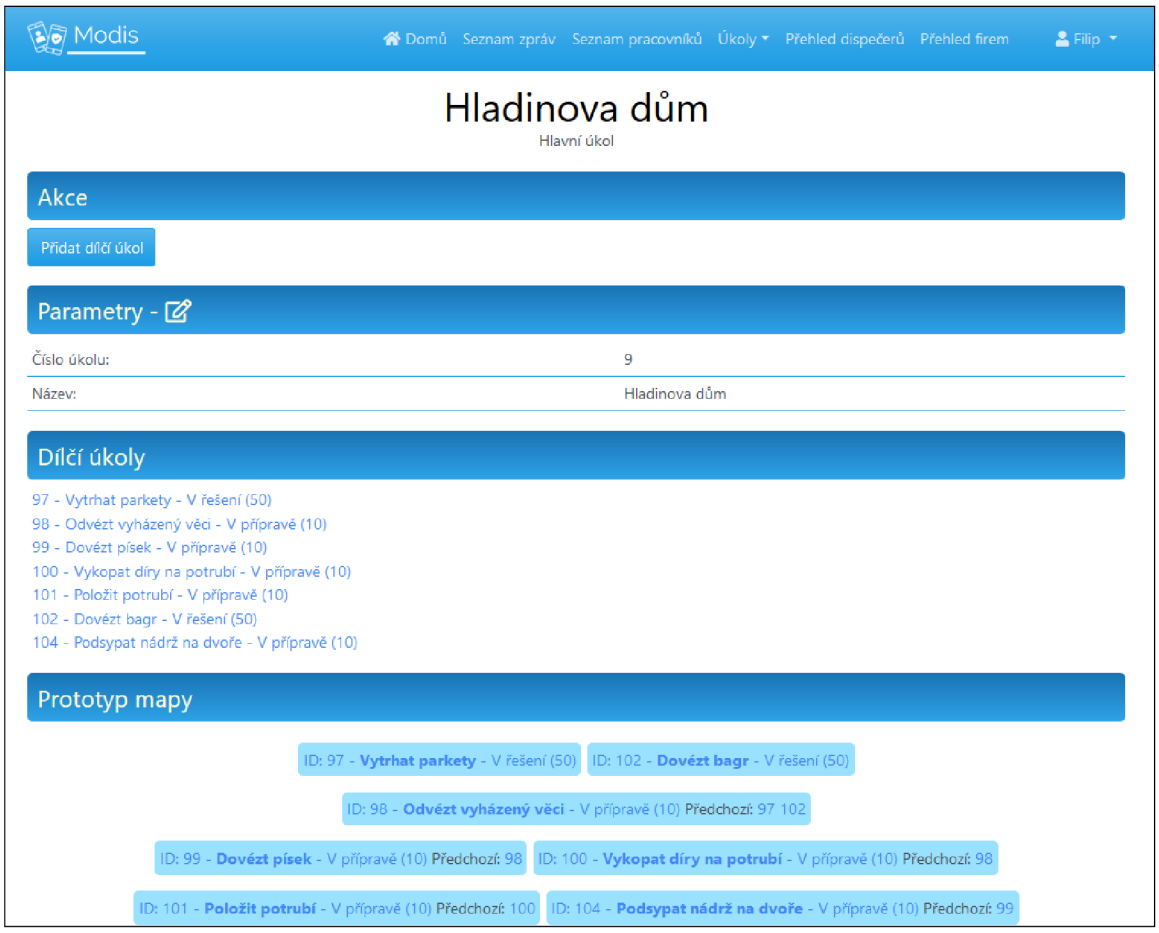
[8] **Material Design** – https://material.io/design



Obrázek 4.8: **Detail přiřazeného úkolu** – v horní části této obrazovky se nachází tlačítka, jejichž dostupnost se mění na základě stavu úkolu a toho, zda je přiřazen do hlavního úkolu. Níže je přehled parametrů, které může dispečer pomocí ikony editovat. Ve spodní části obrazovky se nachází seznam předcházejících a následujících úkolů.



Obrázek 4.9: **Detail pracovníka** – v horní části obrazovky jsou tlačítka pro možné operace. Uprostřed se nachází seznam atributů uživatele, které může dispečer upravit pomocí příslušné ikony. Ve spodní části obrazovky se nachází úkoly přiřazené uživateli a u nich tlačítka na změnu stavu a jejich odebrání.



Obrázek 4.10: **Detail hlavního úkolu** – v horní části obrazovky v sekci „Akce“ je tlačítko pro přidání dílčího úkolu (přiřazený úkol). Níže, v sekci parametry, je tabulka parametrů úkolu. Pod nimi se nachází seznam dílčích úkolů a úplně dole je prototyp mapy zobrazující návaznost dílčích úkolů.

fonu a následné „posouvání“ obrazovky. Stavy úkolů jsou odlišeny různým barevným označením v aplikaci. Toto označení je implementováno jako úzký proužek na levé straně úkolu. Implementace označení je tak minimalistická a působí čistým dojmem a zároveň jsou úkoly od sebe dobře rozeznatelné. Výsledný vzhled seznamu úkolů je zobrazen na obrázku 4.11.

Detail úkolu

Stejná komponenta (SingleChildScrollView) je použita i při výpisu vlastností na obrazovce detailu úkolu. V případě, že má úkol název a popis maximální umožněné délky a uživatel má v systému telefonu nastavené zvětšení veškerého písma, tak se může stát, že obrazovka telefonu opět nebude stačit. Oproti návrhu této obrazovky je zde přidáno tlačítko pro změnu stavu. Tlačítko je na na obrazovce přehledu úkolů i na detailu úkolu z důvodu ulehčení práce pracovníkovi. Ten může změnit stav úkolu po zapnutí aplikace v seznamu všech úkolů nebo v případě, že má otevřený detail úkolu, nemusí přepínat zpět do seznamu. Bylo zde ale nutné vyřešit pozici tlačítek pro změnu stavu úkolu a kontaktování dispečera. Při vložení tlačítek na konec posuvného seznamu by tlačítka v extrémních případech mohla být buď moc nahoře, pod seznamem vlastností, nebo v opačném případě až pod spodní hranou obrazovky, kde by nebyla vidět a bylo by nutné obrazovku posunout, aby se k nim uživatel dostal. Proto byla tlačítka přichycena ke spodní hraně obrazovky a plocha posuvného okna pro přehled vlastností úkolu byla omezena. Při případném posouvání obrazovky detailu úkolu nahoru a dolů tlačítka zůstávají ve stejné poloze ve spodní části obrazovky. Tlačítka se liší podle stavu, v jakém se úkol nachází. Pokud je úkol ve stavu „V pořadí (30)“, jsou všechna tlačítka na jeho detailu skryta. Pokud je úkol ve stavu „V řešení (50)“, pak je na jeho detailu tlačítko pro dokončení a tlačítko pro kontaktování dispečera. Pokud je úkol ve stavu „Dokončen (70)“, pak je na jeho detailu zobrazeno pouze tlačítko pro znovu aktivování úkolu. Detaily úkolu jsou na obrázku 4.11.

4.4 Přihlášení do mobilní aplikace

Pro přihlášení do mobilní aplikace potřebuje pracovník znát své přihlašovací jméno a heslo. Tyto údaje zadá dispečer při vytváření účtu pracovníka. Po zadání přihlašovacích údajů se provede autentizace těchto údajů pomocí API webové aplikace. Pokud jsou přihlašovací údaje validní, pak webová aplikace vygeneruje autentizační token pro příslušného pracovníka a odešle ho jako odpověď na požadavek schválení přihlašovacích údajů.

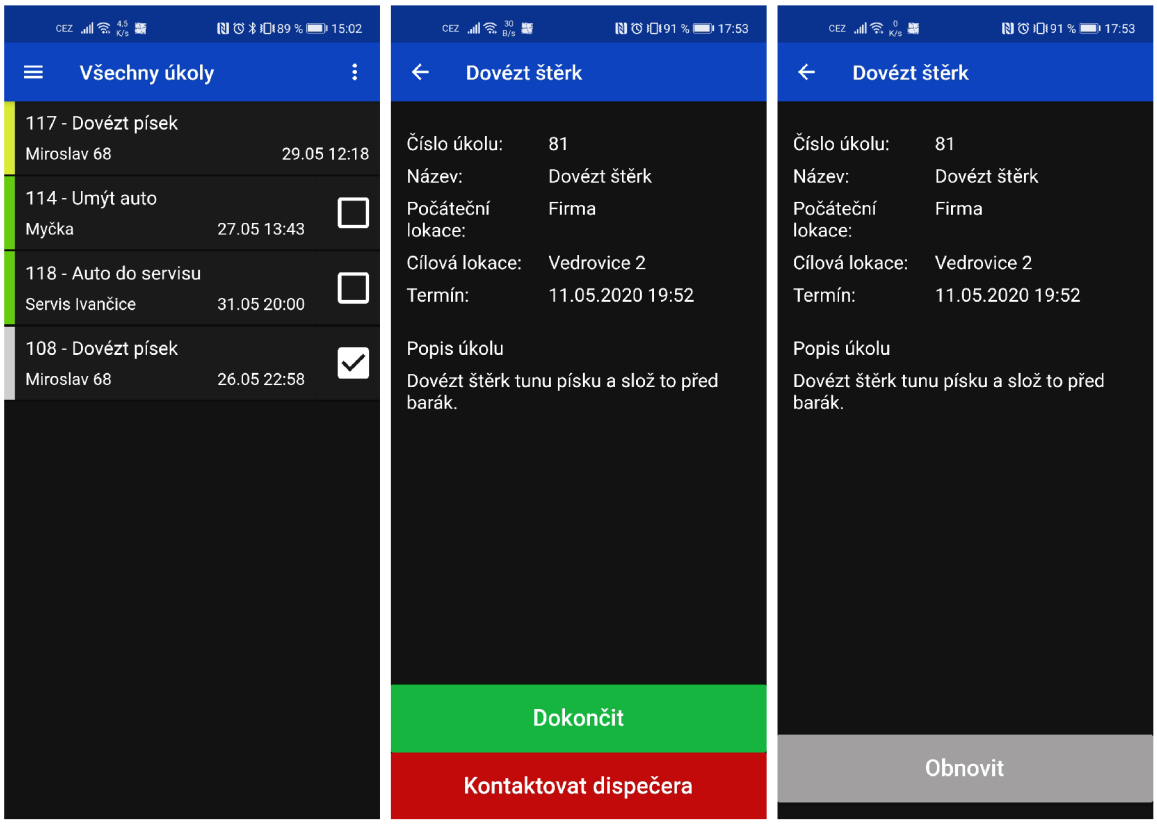
Po schválení přihlašovacích údajů a obdržení autentizačního tokenu si mobilní aplikace uloží ID uživatele, jeho autentizační token a expirační datum tokenu, po kterém mobilní aplikace uživatele odhlásí a bude požadovat opětovné přihlášení. Tato data jsou do mobilní aplikace uložena pomocí `Shared Preferences`[9].

Při každém dalším požadavku na API webové aplikace se mobilní aplikace autentizuje obdrženým tokenem.

Při odhlášení se provede smazání uživatelských dat uložených v mobilní aplikaci a uživatel je odhlášen.

Pro přihlašování pomocí tokenu existují různé knihovny, pomocí kterých jsem se pokoušel vyřešit autentizaci použití API. Tyto knihovny se mi však nepodařilo správně implementovat a tudíž nefungovaly tak, jak měly. A rychlejší než zkoumat, kde je problém, bylo napsat vlastní obdobu těchto knihoven.

[9] **Shared Preferences** – Interface pro ukládání uživatelských dat mobilních aplikací.



Obrázek 4.11: **Přehledy úkolů v mobilní aplikaci** – Na obrázku vlevo je obrazovka s přehledem úkolů v mobilní aplikaci. Na obrázku uprostřed je detail úkolu, který je ve stavu „V řešení (50)“. Na obrázku vpravo je detail úkolu, který je ve stavu „Dokončen (80)“.

4.5 Kontaktování dispečera z mobilní aplikace

Pracovník musí mít možnost rychle a snadno kontaktovat dispečera v případě, že se vyskytne problém při plnění úkolu. Pro tento případ je na detailu úkolu (obrázek 4.11) tlačítko, které otevře dialogové okno s možností zadání zprávy dispečerovi. Pokud pracovník do pole nezadá žádnou zprávu, pak se dispečerovi odešle zpráva vázána na daný úkol s textem, že pracovník nezadal žádnou zprávu. V takovémto případě se předpokládá, že dispečer pracovníkovi zavolá. Pokud pracovník zprávu zadá, pak se text odešle jako tělo zprávy. Zpráva se do webové aplikace odešle pomocí přístupu na její API. Dialogové okno pro kontaktování dispečera je zobrazeno na obrázku 4.12.

4.6 Notifikace

Jak již bylo zmíněno, aplikace upozorňují uživatele na úkoly blížící se termínu dokončení a úkoly, kterým termín dokončení již vypršel. Webová aplikace také upozorňuje dispečera na příchozí zprávu a mobilní aplikace pracovníka upozorní v případě, že mu dispečer odešle nový přiřazený úkol.

4.6.1 Notifikace webové aplikace

Kontrola událostí pro zobrazení notifikace ve webové aplikaci se děje pomocí Javascriptového kódu běžícího na pozadí webové aplikace. Funkce po určitém časovém intervalu opakovaně spustí kontrolu termínů úkolů tím, že provede fetch na adresu webové aplikace, která zkontroluje termíny úkolů. Ta případně vrátí obsah, který se vypíše do zobrazeného notifikačního dialogu. Dialog je zobrazený na obrázku 4.2 a 4.3.

4.6.2 FCM – Notifikace do mobilní aplikace

Mobilní aplikace obsahuje notifikace na nově přidělený úkol, úkoly blížící se termínu dokončení a úkoly, kterým vypršel termín dokončení. Notifikace před vypršením úkolů slouží k upozornění pracovníka v případě, že je úkol splněn, ale pracovník zapomněl potvrdit jeho dokončení. Dispečer se pak zbytečně nemusí zaobírat úkolem.

Notifikace do mobilního telefonu jsou implementovány pomocí FCM. Tato služba je blíže popsána v kapitole 2.4.

V aplikaci je využívána možnost odesílání notifikací právě na jedno zařízení. V okamžiku, kdy uživatel spustí mobilní aplikaci, tak mobilní aplikace požádá o přidělení tokenu služby FCM. Tato žádost se děje přímo v mobilní aplikaci pomocí volání inicializační metody `configure` této služby implementované v její knihovně `firebase_messaging.dart` na proměnnou typu `FirebaseMessaging`. Po provedení inicializace je zařízení přihlášeno k odběru notifikací a příslušný identifikační token je dostupný v dané proměnné po zavolání metody `getToken`.

Tento FCM token se do databáze webové aplikace uloží v momentě, kdy se uživatel přihlásí do mobilní aplikace a přihlášení je úspěšné. Ukládá se jako záznam obsahující hodnotu tokenu a odkaz na uživatele, kterému tento token patří. Mobilní aplikace zažádá o uložení tokenu do databáze. Webová aplikace provede kontrolu unikátnosti FCM tokenu a v případě konfliktu starý token vymaže. Tato kontrola je z důvodu, že pokud uživatel smaže data aplikace nebo aplikaci odinstaluje bez předchozího odhlášení, může nastat konflikt tokenu. Znamenalo by to, že by se na příslušný telefon notifikace odesílaly tolikrát, kolikrát by byl duplicitní token uložen v databázi.

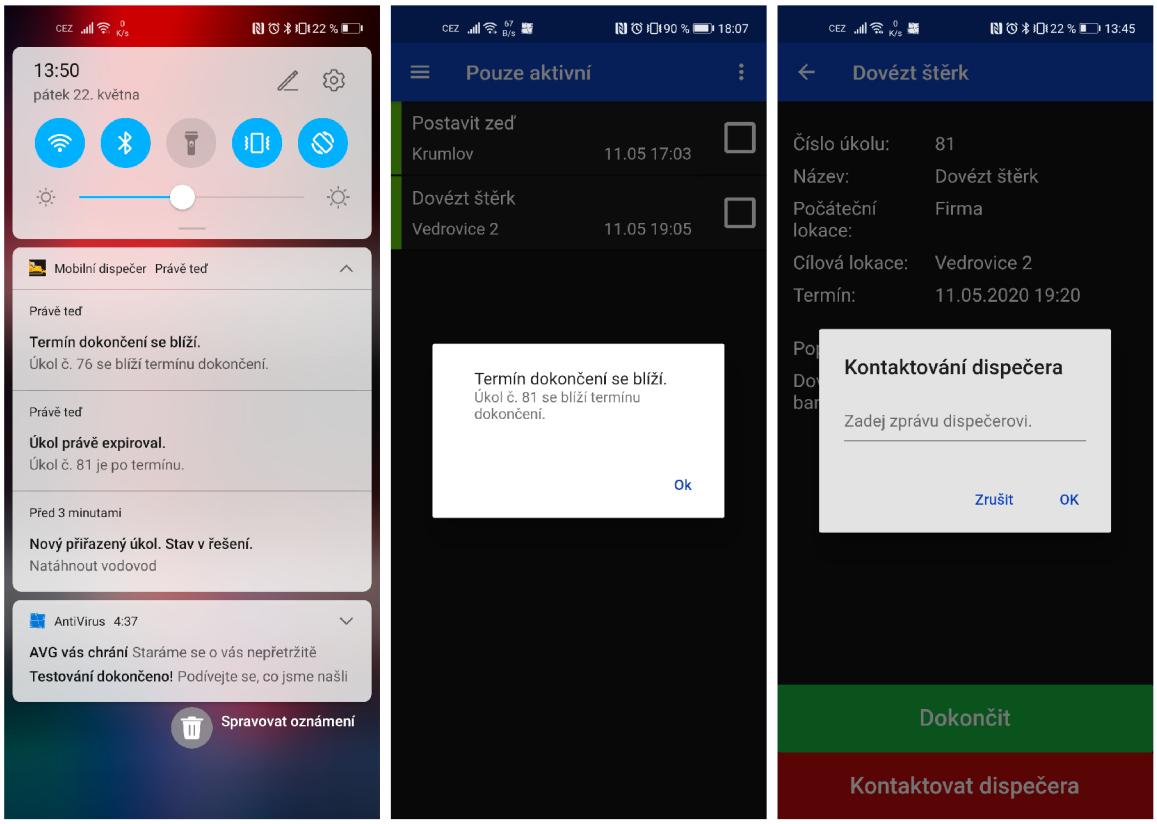
Při odhlašování z mobilní aplikace je před samotným odhlášením proveden požadavek mobilní aplikace na webovou aplikaci na smazání příslušného tokenu z databáze. Zaručí se tím, že po odhlášení z aplikace uživateli nebudou chodit notifikace.

Díky výše popsanému způsobu ukládání tokenu je možné, aby byl jeden uživatel přihlášen na více zařízeních současně a notifikace chodily na všechna zařízení.

Notifikace při zavřené mobilní aplikaci či aplikaci na pozadí, jsou nastaveny automaticky. Příchozí zpráva obsahuje položku `title`, která se použije jako titulek notifikace a položku `body`, která se použije jako tělo zprávy. Takto vytvořená notifikace je zobrazena na obrázku 4.12. Notifikace při otevřené aplikaci je nutno nastavit. Při inicializaci třídy FCM se volá metoda `configure`, kde je možné nastavit akci při příchozí notifikaci, když je aplikace otevřená. Je to akce `onMessage`, kde se volá metoda `handleNotificationMsg` ze třídy `notification_manager.dart`, která na základě příchozí notifikace zobrazí dialog. Takto přijatá notifikace je zobrazena na obrázku 4.12

Ve webové aplikaci se odeslání notifikace do mobilního zařízení provádí pomocí použití metody `sendNotificationsToUsers`, která je součástí modelu `FcmModel` a je popsána v kapitole 4.7.2.

Samotná kontrola na podnět pro odeslání notifikace probíhá ve webové aplikaci. Není k tomu potřeba, aby byl dispečer aktuálně u PC, jak je tomu v případě kontroly notifikací



Obrázek 4.12: **Notifikace mobilní aplikace a kontaktování dispečera** – Na obrázku vlevo je způsob zobrazení notifikací, pokud aplikace není spuštěna. Na obrázku uprostřed je způsob zobrazení notifikací, pokud je aplikace spuštěna. Na obrázku vpravo je dialogové okno sloužící pro kontaktování dispečera.

pro dispečera v kapitole 4.6.1. Symfony nepodporuje způsob automatického spouštění kódu na pozadí serveru paralelně s webovou aplikací. Spuštění kontroly je vyřešeno přístupem na konkrétní adresu webové aplikace. Při přístupu se provede kontrola a odešlou se případné notifikace. Pravidelný přístup na adresu aplikace zajišťuje služba Cronjobs[10], která každých 5 minut provede přístup. Vyjímku tvoří notifikace na nově přiřazené úkoly. Tyto notifikace se odesílají ve chvíli, kdy dispečer klikne na tlačítko odeslat a tím pádem není nutná pravidelná kontrola.

4.7 Implementace vybraných tříd

Každý celek aplikace má své vlastní kontrolery a své vlastní modely. Kontrolery slouží k definování kódu, který se spustí po přístupu na určitou adresu aplikace. Jsou implementovány jako třídy a jejich jednotlivé metody obsluhují jednotlivé stránky. Jsou dvě základní použití kontrolerů.

Prvním je načtení a zpracování dat a poskytnutí těchto dat TWIG šabloně. Tato šablona na základě dat vygeneruje html dokument, který kontroler předá a uživatel ho vidí ve svém prohlížeči.

[10] **Cronjobs** – cron-job.org

Dalším možným použitím kontroleru při přístupu na určitou stránku je získání dat, případně úprava dat v databázi. Na takovou adresu se přistupuje pomocí fetche z JavaScriptového kódu webové aplikace, nebo fetche z mobilní aplikace. Výsledkem takového volání není zobrazená html stránka, ale data ve formátě JSON, případně potvrzení o úspěšné operaci. Obecný popis kontrolerů v rámci modelu MVC se nachází v kapitole 2.2.1.

4.7.1 Jednotlivé kontrolery a jejich stručný popis

V této kapitole je seznam jednotlivých kontrolerů s jejich krátkým popisem. U jednotlivých kontrolerů jsou vypsány a popsány i vybráné metody těchto kontrolerů.

- **CompaniesController** – slouží ke správě jednotlivých firem v aplikaci. Tyto firmy slouží pro oddělení uživatelů a jejich úkolů a pracovníků v aplikaci. Kontroler obsahuje metody pro:
  - zobrazení seznamu firem,
  - zobrazení detailu třídy.
- **EmployeesController** – slouží ke správě pracovníků. Obsahuje metody pro:
  - vypsání seznamu pracovníků jedné firmy,
  - zobrazení detailu pracovníka,
  - metodu pro smazání pracovníka.
- **EmployeeTaskController** – tento kontroler je jeden z nejobsáhlejších a nejsložitějších. Obsahuje metody pro správu přiřazených úkolů. Konkrétní použití metod je:
  - vypsání seznamu přiřazených úkolů patřících pod určitou firmu,
  - vypsání seznamu přiřazených úkolů patřících pod určitou firmu a ve stavu „Uzavřen (80)“,
  - zobrazení detailu přiřazeného úkolu,
  - metodu pro smazání přiřazeného úkolu,
  - metody pro změnu stavu úkolů,
  - odstranění přiřazeného úkolu z hlavního úkolu,
  - získání úkolů blížících se termínu a úkolu po termínu dokončení pro upozornění dispečera.
- **HomepageController** – tento kontroler má pouze jedinou funkcionalitu. A sice zobrazení pracovníků a jejich přiřazených úkolů ve stavech „V pořadí (30)“ a „V řešení (50)“.
- **MainTaskController** – účelem tohoto kontroleru je spravovat hlavní úkoly. Obsahuje metody pro
  - vypsání seznamu hlavních úkolů patřících jedné firmě,
  - vypsání hlavních úkolů patřících jedné firmě ve stavu „Uzavřen (80)“,
  - zobrazení detailu hlavního úkolu,
  - uzavření hlavního úkolu.

- **MessagesController** – spravuje zprávy, které odesílá pracovník dispečerovi z mobilní aplikace. Obsahuje metody pro
  - zobrazení seznamu zpráv, které nejsou vyřešeny,
  - označení zprávy jako vyřešené,
  - metodu pro získání nevyřešených zpráv pro jejich zobrazení ve vyskakovacím okně.
- **RestApiController** – tento kontroler se stará o veškeré přístupy na API webové aplikace. Obsahuje metody pro
  - ověření přihlašovacího jména a hesla pracovníka z mobilní aplikace a případné vygenerování a odeslání autentizačního tokenu pro webovou aplikaci,
  - získání přiřazených úkolů jednomu pracovníkovi na základě jeho autentizačního tokenu,
  - metodu pro provedení kontroly termínů úkolů a případné odeslaní notifikace do mobilní aplikace,
  - metody pro změnu stavu úkolu za nutnosti autentizace pomocí tokenu,
  - metodu pro odeslání zprávy dispečerovi,
  - registraci FCM tokenu ve webové aplikaci,
  - smazání FCM tokenu ve webové aplikaci.
- **SecurityController** – kontroler sloužící pro přihlašování a odhlašování k webové aplikaci.
- **TasksController** – slouží pro správu šablon úkolů. Obsahuje metody pro
  - zobrazení seznamu šablon úkolů patřících jedné firmě,
  - smazání šablony úkolu,
  - zobrazení detailu šablony.
- **UsersController** – spravuje uživatele webové aplikace. Obsahuje metody pro
  - zobrazení seznamu všech uživatelů,
  - smazání uživatele,
  - zobrazení detailu uživatele.

4.7.2 Jednotlivé modely a jejich stručný popis

Modely obsahují aplikační logiku. Zpracovávají požadavky, provádí různé operace nebo vrací požadovaná data. V této kapitole je jejich seznam a stručný popis nejdůležitějších částí.

- **ApiModel** – tento model poskytuje funkcionalitu pro Rest API sloužící ke komunikaci mobilní a webové aplikace. Obsahuje metody pro:
  - Nalezení uživatele v databázi dle přihlašovacího jména nebo autentizačního tokenu. Tyto metody se využijí při ověřování přihlašovacích údajů nebo například při změnách stavů úkolů z mobilní aplikace či jejich načtení do mobilní aplikace.

  - Správu autentizačního tokenu webové aplikace. Při vytváření tokenu se pomocí funkce `random_bytes` vygeneruje binární posloupnost o délce 20. Tato posloupnost ovšem může obsahovat také znak `\0`. Tento znak může při práci s řetězcem způsobit problémy, proto se celý řetězec ještě „zahešuje“ pomocí funkce `md5`.
  - Metody pro nalezení úkolů blížících se termínu a po termínu. Tyto metody využívají `FcmModel`, pomocí kterého odesílají notifikace do mobilních zařízení. V těchto metodách je třeba dbát na nastavení správné časové zóny. Časová zóna se implicitně nastaví dle časového pásma, ve kterém se nachází server, na kterém běží aplikace.

- **EmployeeModel** – tento model obsahuje metodu pro získání všech pracovníků jedné společnosti a metodu pro získání šablon úkolů jedné společnosti.

- **EmployeeTaskModel** – model zabezpečuje operace nad přiřazenými úkoly. Jsou zde například metody pro:
  - Získání přiřazených úkolů pracovníka bez nepotřebných atributů pro odeslání do mobilní aplikace.
  - Nastavení stejného hlavního úkolu u přiřazených úkolů.
  - Kontroly a získávání úkolů pro nastavování předchozího a následujícího přiřazeného úkolu. Tyto kontroly jsou nutné proto, aby nevznikaly cyklické závislosti ve vazbách úkolů. Kontroly se provádějí rekurzivním získáním všech navázaných úkolů a jejich následným procházením a porovnáváním.
  - Vyplnění přiřazeného úkolu při jeho vytváření z šablony úkolu.
  - Metody pro změnu stavu úkolu a kontroly předcházející těmto změnám. Metody pro změnu stavu případně obsahují odesílání notifikací do mobilních zařízení.
  - Metody pro získání úkolů blížících se termínu dokončení a úkolů po termínu dokončení pro notifikace ve webové aplikaci.

- **FcmModel** – tento model obsahuje metody pro odeslání notifikací do mobilních zařízení. Metoda požaduje zadat přiřazený úkol, titulek notifikace a její text. Poté se provede získání všech tokenů zařízení, ve kterých je přihlášen uživatel, kterému je přiřazen úkol. Na všechna tato zařízení se následně odešle notifikace.

- **MainTaskModel** – obsahuje metody pro práci s hlavními úkoly. Jsou zde metody pro přidání přiřazeného úkolu do hlavního úkolu a zároveň všech jeho předcházejících nebo následujících úkolů. Zajímavá je metoda, která vytváří z podúkolů hlavního úkolu pole polí úkolů pro jejich zobrazení v grafu. Tento proces se provádí pomocí rekurze, kdy se zjišťuje nejdelší cesta úkolu ke kořenu úkolu. Tím se zjistí úroveň, na které se daný úkol nachází a podle toho se vloží na příslušný index pole.

- **MessageModel** – tento model obsahuje metody pro získání nepotvrzených zpráv. Metody jsou dvě. Jedna slouží k jejich výpisu ve webové aplikaci a druhá metoda z úkolů vytváří řetězec s HTML kódem pro vypsání zpráv v notifikačním okně aplikace.

- **TaskModel** – tento model slouží k práci se šablonami úkolů. Obsahuje metodu pro získání všech šablon úkolů patřících jedné firmě.

4.8 Komunikace webové a mobilní aplikace

Ke komunikaci webové a mobilní aplikace se používá Rest API webové aplikace. Mobilní aplikace odesílá fetch požadavky na API pomocí knihovny `http.dart`. Proces komunikace webové a mobilní aplikace při přihlašování je popsán v kapitole 4.4. Komunikace pro správu FCM tokenu je popsána v kapitole 4.6.2.

Webová aplikace na základě ověření přihlašovacího jména a hesla mobilní aplikaci odešle token, pomocí kterého se aplikace bude při každém požadavku identifikovat.

Pro získání úkolů mobilní aplikace pomocí tokenu požádá přes API o úkoly. Po ověření jsou jako odpověď navráceny úkoly uživatele.

Pro přehlednost se do mobilní aplikace odesílají pouze úkoly ve stavech „V pořadí (30)“, „V řešení (50)“ a „Dokončen (70)“. Úkoly ve stavu „Dokončen (70)“ se do aplikace odešlou pouze za podmínky, že není překročen jejich požadovaný termín splnění. Je to z důvodu, aby pracovník mohl v případě potřeby vrátit úkol ve stavu „Dokončen (70)“ do stavu „V řešení (50)“ a zároveň, aby se do aplikace zbytečně neodesílaly dokončené úkoly po termínu.

Požadavek o změnu stavu úkolu z mobilní aplikace probíhá také přístupem na API webové aplikace. Mobilní aplikace se opět autentizuje obdrženým tokenem. Na základě obdržené odpovědi, zda se operace zdařila, změní stav úkolu v aplikaci bez nutnosti opětovného načítání všech úkolů.

4.9 Publikování aplikace

Aplikace je umístěna na Heroku[11]. To poskytuje vhodné nástroje pro budoucí umístění mé aplikace i pro vývoj aplikace. Při použití základní verze je služba zcela zdarma. Jediné menší omezení, které není ideální pro používání aplikace je mechanismus, který aplikace na serveru uspává po delší době neaktivity a následující první přistup po uspání trvá pár sekund, než se běh aplikace obnoví. Pro studenty je ale k dispozici zdarma roční vylepšení hostingu, které poskytuje větší výkon a odstraňuje omezení v podobě uspávání aplikace.

Na heroku jsem si založil dvě aplikace. Jednu, na které běžela stabilní aplikace, bylo nastaveno produkční prostředí a bylo možné, aby zde aplikaci testovali uživatelé. Druhou pro účely vývoje. Zde bylo nastaveno vývojové prostředí. I když jsem aplikaci vyvíjel v lokálním prostředí, tak ji bylo potřeba otestovat i na serveru. Některé funkcionality se totiž lišily od toho, kde aplikace běžela a bylo třeba upřesnit nastavení.

Nahrávání nových verzí na server jsem realizoval pomocí GITu. Jakmile se vytvoří nový commit, tak pomocí příkazu `git push dev dev:master` je na server nahrána nová verze. "dev"označuje heroku aplikaci, na kterou chci nahrávat. "dev:master"značí větev lokálního GITu a větev GITu v aplikaci.

[11] **Heroku** – cloudová platforma PaaS. heroku.com

4.10 Zabezpečení

Při vývoji aplikace bylo dbáno na její bezpečnost.

4.10.1 Webová aplikace

Symfony obsahuje funkcionalitu pro implementaci přihlašování. Provede se to příkazem `php bin/console make:auth` a následným upřesněním několika parametrů, dle kterých je vytvořeno přihlašování. Přihlášení ovládá `SecurityControler`. V souboru `login.html.twig` je možné upravit vzhled přihlašovací obrazovky.

Dále je nutné v souboru `security.yaml` nastavit přístup jednotlivým adresám aplikace. Jediná stránka v aplikaci, která je zpřístupněna nepřihlášenému uživateli je přihlašovací obrazovka. Pokud se uživatel pomocí odkazu, nebo přepsáním adresy v prohlížeči dostane na jakoukoliv jinou stránku a není přihlášen, tak je přesměrován na přihlašovací obrazovku. V tomto souboru je také nastaveno jaké role musí mít uživatel přiděleny, aby se dostal na určité stránky. Aplikace implementuje dvě role – `ROLE_USER` a `ROLE_ADMIN`. `ROLE_USER` má každý uživatel přihlašený k aplikaci automaticky. Tato role neposkytuje uživateli žádná práva navíc. `ROLE_ADMIN` označuje správce aplikace. S touto rolí má správce přistup ke stránkám jako správa firem, nebo správa dispečerů.

V souboru `security.yaml` je dále nastaveno vyžadování použití protokolu https pro všechny stránky aplikace. Pokud se tedy uživatel pokusí stránku načíst pomocí protokolu http, automaticky je přesměrován na stejnou adresu s použitím https.

Vzhledem k tomu, že je aplikace víceuživatelská, bylo nutné vyřešit oddělení dat jednotlivých uživatelů. Aplikace data načítá dle přihlášeného uživatele. Některé stránky ale informaci o tom, jaká data mají zobrazit, berou z adresy zadané ve webovém prohlížeči. To znamená, že pokud by uživatel zobrazil detail svého úkolu a poté přepsal v prohlížeči v adrese číslo úkolu, mohl by zobrazit detail cizího úkolu. Veškeré možnosti takového nabourání se do aplikace jsou ošetřeny a v případě neoprávněného přístupu je uživatel přesměrován (například z detailu úkolu na seznam všech úkolů).

V aplikaci není možnost registrace. Vytvoření nového uživatele v databázi je možné pouze pomocí rozhraní aplikace, pokud má přihlášený uživatel (správce) přidělenou roli `ROLE_ADMIN` nebo případně přímým přístupem do aplikace. Na přihlašovací obrazovce je zobrazený kontakt, kde si případný budoucí uživatel může zažádat o vytvoření účtu.

4.10.2 Mobilní aplikace

Pro přístup do mobilní aplikace je nutné se přihlásit. Každý uživatel webové aplikace, v praxi dispečer, má možnost vytvořit účty pro své pracovníky, pomocí kterých se budou moci do mobilní aplikace přihlásit.

Distribuce mobilní aplikace je zabezpečena pomocí služby Obchod Play. Tato služba implementuje podepisování aplikací speciálním klíčem. Není tedy možné, aby útočník, i v případě, že získá přístup k mému účtu na Google Play, byl schopen vyměnit aktuální dostupnou verzi aplikace za podvodnou.

Ověření přihlašovacího jména a hesla probíhá pomocí webové aplikace. Proces komunikace mobilní a webové aplikace je popsán v kapitole 4.4.

4.11 Testování

Průběh testování aplikace neproběhl podle mých představ. Počítal jsem s tím, že firma, která o aplikaci projevila zájem, bude ochotna testovat aplikaci v rámci jejího iterativního vývoje. Stálo by to nějaké úsilí a čas, protože by se muselo pracovat s aplikací, která nemusí vždy fungovat a má omezenou funkcionalitu. Dosáhlo by se tím ale mnohem lepšího výsledku, který by si mohla firma více přizpůsobit dle svých potřeb. S firmou jsem komunikoval přes prostředníka Ing. Aleše Potůčka, který zjišťoval požadavky firmy a řešení se mnou konzultoval. Ten také aplikaci testoval v průběhu vývoje a simuloval reálné nasazení.

Nejrozsáhlejší a nejužitečnější testování proběhlo při vyvíjení aplikací doma. Aplikaci jsem nejvíce konzultoval se svým otcem, který už nějaké zkušenosti s vývojem aplikací má. Při testování a konzultování vytvořeného řešení tedy dokázal vhodně otestovat a zhodnotit vytvořenou část a případně navrhnout lepší řešení. Testování probíhalo pravidelně už od začátku samotného vývoje. Vždy, když byla vytvořena nová funkcionalita nebo byl upraven nějaký větší celek aplikace hodný kontroly, bylo provedeno testování a vyhodnoceno aktuální řešení a případně navrhnuty možné úpravy pro zlepšení aplikací. Nejprve byly prováděny triviální úkony pro zjištění funkcionality jednoho tlačítka v různých možných situacích. Postupně, jak se aplikace rozšiřovaly, byly prováděny komplexnější úkony pro zjištění celkové funkčnosti aplikace.

Testování prováděl i Ing. Aleš Potůček. Jeho testování bylo užitečné z pohledu nezávislého testujícího, který ví, co se od aplikací očekává, ale nezná dopodrobna aktuálně vytvořené řešení. Díky tomu bylo možné zhodnotit například, zda je rozložení prvků v aplikaci vhodné a jestli dělají to, co by se od nich na první pohled očekávalo, nebo jestli jsou postupy pro přidělování úkolů v aplikaci intuitivní a jasné.

V pozdější fázi vývoje, kdy už aplikace získávala na rubustnosti, jsem začal tlačit na testovací nasazení do firmy. Tehdy jsem zjistil, že firma nemá zájem aplikaci testovat v rámci vývoje a požaduje již hotovou aplikaci. Bylo tedy nutné dále testovat aplikaci v náhradním prostředí, dokud nebude aplikace zcela použitelná pro potřeby firmy. Testování v simulovaném prostředí bylo užitečné a mělo dobré výsledky, ale bylo třeba zapojit i méně znalé a zdatné uživatele, kteří ještě lépe nasimulují použití v praxi. Proto jsem zapojil do testování i další členy rodiny a na základě výsledků několika denního testování doladil aplikaci. Tito testeři navíc neměli zkušenost s předchozí prací s aplikacemi. Což bylo užitečné z pohledu testování použitelnosti, které je popsáno v knize, kterou napsal Steve Krug [4].

Postupným dalším vývojem a testováním se aplikace dostala do fáze, kdy již byla připravená na nasazení do firmy a konečně i otestování v reálném prostředí. Toto testování zařídil Ing. Aleš Potůček, který již dříve s firmou komunikoval, testování domluvil a provedl nasazení. Výsledky testování přinesly očekávaný výsledek. A sice, že aplikace funguje, jak má a splňuje funkcionalitu popsanou v požadavcích a je vhodným přínosem pro práci v dané oblasti. Firma však požaduje rozšíření pro ještě větší zjednodušení práce, které už z důvodu pozdního nasazení aplikace do testování ve firmě a blížícího se termínu odevzdání není možné udělat v rámci bakalářské práce. Jedná se například o rozšíření formuláře při přidělování úkolu, aby mohl dispečer jednoduše naklikat materiál a jeho množství, které je třeba někam přepravit, dále pak například zahrnutí objednávky, fakturace i platby do posloupnosti stavů úkolů. Tyto změny jsou zahrnuty do dalšího plánovaného vývoje. Vzhledem k univerzálnosti navrženého řešení jsou aplikace dalšímu rozšíření otevřené.

Celkový vývoj a testování také zkomplikovala epidemie nemoci COVID-19, kvůli které jsem se osobně nesetkal s lidmi ve firmě a musel spoléhat pouze na komunikaci pomocí prostředníka.

Kapitola 5

Závěr

Na základě požadavků stavební firmy, které jsem zjednodušil a zobecnil tak, aby byla zachována požadovaná funkčnost, ale aplikace byla zároveň univerzální a dala se použít i jinde než ve stavebnictví, jsem navrhl a vytvořil webovou a mobilní aplikaci pro zadávání a potvrzování úkolů.

V úvodu tvorby aplikací jsem vyhledal, prozkoumal a zhodnotil vhodná existující řešení pro danou problematiku. V rámci průzkumu jsem nenašel aplikaci, která by splňovala všechny potřebné požadavky. Poté jsem vyhledal a zvolil vhodné technologie pro návrh a implementaci aplikace. Na základě výsledků omezeného testování jsem aplikaci iterativně vyvíjel a zlepšoval.

Z důvodu epidemie COVID-19 neproběhlo testování v takovém rozsahu, v jakém bylo naplánováno na začátku tvorby práce. Osobní setkání s testery aplikace a nasazení do praxe bylo uskutečněno pouze v omezené míře. Aplikace byla však v průběhu vývoje testována simulací reálného použití a tím se dosáhlo vytvoření použitelné a užitečné aplikace. Tento fakt byl potvrzen při nasazení vyladěné aplikace do testovaní v praxi, kde byla ověřena její funkčnost.

Zadání práce bylo splněno. Obě aplikace jsou naprosto dostačující pro použití v zadané oblasti. Je však možné aplikace dále vylepšovat a dosáhnout tím ještě větší efektivity. Mezi hlavní plánovaná budoucí vylepšení patří například šablony více navazujících úkolů, mapa s aktuální pozicí pracovníka a cílem cesty pracovníka, vylepšení grafu návaznosti úkolů, zahrnutí kompletní objednávky včetně fakturace nebo rozšířené možnosti formulářů při vytváření úkolů. Webová aplikace je dostupná na adrese `mobilniDispecer.cz`. Mobilní aplikace je k dispozici v aplikaci Obchod play pod názvem „Mobilní dispečer“[1]. V rámci dalšího vývoje je možné vytvořit verzi aplikace pro zařízení Apple. V budoucnu se budu snažit nasadit aplikaci do více firem. K aplikacím jsem vytvořil plakát a video, které jsem umístil na server Youtube[2].

S vývojem ve frameworku Symfony a frameworku Flutter jsem spokojený. Tvorba aplikací byla pohodlná a nenarazil jsem na nějaké větší omezení či nedostatky těchto frameworků. Aplikace jsou v praxi přínosem díky zlepšení efektivity práce a ušetření zdrojů a uživatelé jsou s nimi spokojeni.

[1] `https://play.google.com/store/apps/details?id=com.herokuapp.modis.modil_app5`
[2] `https://www.youtube.com/watch?v=fUzxktSB9e0`

Literatura

[1] Gasston, P. *Moderní web*. 2. vyd. Computer Press, 2015. ISBN 978-80-251-4345-2.

[2] Hogan, B. P. *HTML5 and CSS3: Level Up with Today's Web Technologies*. 2. vyd. Pragmatic Bookshelf, 2013. ISBN 978-1-937-78559-8.

[3] Krug, S. *Don't Make Me Think: A Common Sense Approach to Web Usability, 2nd Edition*. 2. vyd. New Riders, 2005. ISBN 978-0-321-34475-8.

[4] Krug, S. *Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability Problems*. 1. vyd. New Riders, 2009. ISBN 978-0-321-65729-9.

[5] Schwarzmüller, M. *Flutter & Dart - The Complete Guide [2020 Edition]* [online]. 2020 [cit. 2019-04-22]. Dostupné z: https://www.udemy.com/course/learn-flutter-dart-to-build-ios-android-apps/.

[6] Luke Welling, L. T. *Mistrovství - PHP a MySQL*. 1. vyd. Albatros Media a.s., 2017. ISBN 978-8-025-14901-0.

[7] Marsh, J. *UX for Beginners: A Crash Course in 100 Short Lessons*. 1. vyd. O'Reilly Media, Inc., 2015. ISBN 978-14-919-1264-5.

[8] Máca, J. *Lekce 1 - Úvod do Symfony frameworku pro PHP* [online]. 2017 [cit. 2020-05-11]. Dostupné z: https://www.itnetwork.cz/php/symfony/zaklady/uvod-do-symfony-frameworku-pro-php.

[9] Stark, J. *The 10 principles of mobile interface design* [online]. 2012 [cit. 2020-05-11]. Dostupné z: https://www.creativebloq.com/mobile/10-principles-mobile-interface-design-4122910.

[10] Thomas, G. *What is Flutter and Why You Should Learn it in 2020* [online]. 2019 [cit. 2020-05-11]. Dostupné z: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/.

[11] Čápka, D. *Lekce 1 - Úvod do PHP a webových aplikací* [online]. 2013 [cit. 2020-05-11]. Dostupné z: https://www.itnetwork.cz/php/zaklady/php-tutorial-uvod-do-webovych-aplikaci.

[12] Žára, O. *JavaScript*. 1. vyd. Albatros Media a.s., 2015. ISBN 978-8-025-14593-7.