ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Systém pro záznam, generaci a simulaci docházky

Kryštof Woldřich
Softwarové inženýrství a technologie

Květen 2020
Vedoucí práce: Ing. Jiří Fryč

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Woldřich** Jméno: **Kryštof** Osobní číslo: **466201**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro záznam, generaci a simulaci docházky

Název bakalářské práce anglicky:

System for attendence recording, generating and simulation

Pokyny pro vypracování:

Analyzujte agendu spojenou s vytvářením dokumentů docházky.
Analyzujte obecné požadavky kladené na systém pro záznam, generaci a simulaci docházky s ohledem na pracovní úvazky na ČVUT FEL. Před implementací popište případy užití.
Navrhněte a implementujte systém vyhovující výše zmíněným obecným požadavkům a případům užití. Navrhněte vhodné postupy pro testování jednotlivých komponent systému i celého systému. Testy proveďte.

Seznam doporučené literatury:

[1] Morgan, A. and Anthony, S. (2008) Creating a high-performance workplace: A review of issues and opportunities. Journal of Corporate Real Estate 10 (1): 27–39.
[2] Aronoff, S. and Kaplan, A. (1995) Total Workplace Management: Rethinking the Office Environment. Canada: WDL Publications.
[3] Hassanain, M. Analysis of factors influencing office workplace planning and design in corporate facilities. J Build Apprais 6, 183–197 (2010) doi:10.1057/jba.2010.22

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Fryč, katedra ekonomiky, manažerství a humanitních věd FEL

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

Datum zadání bakalářské práce: **13.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

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

III. PŘEVZETÍ ZADÁNÍ

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

. | Datum převzetí zadání

Podpis studenta

Poděkování / Prohlášení

Děkuji své rodině a nejbližším za jejich podporu, díky které jsem se mohl naplno věnovat této práci. Zároveň bych chtěl poděkovat vedoucímu bakalářské práce Ing. Jiřímu Fryčovi za cenné rady a přípomínky, které jsem využil při zpracování této práce.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. 5. 2020

.....

Abstrakt / Abstract

Cílem této práce je zpracovat analýzu a implementovat docházkový software dle zadání. Tato aplikace je určena pro administrativní pracovníky v akademickém prostředí. Řeší problém časově náročného zpracování docházky, kde se data třídí podle zaměstnanců a projektů pro následné další zpracování. Cílem aplikace je zrychlit tento proces a přípravu dat pro třetí strany.

Aplikace umožňuje manuální vkládání a dávkový import dat o zaměstnancích, jejich úvazcích a projektech, na kterých pracují nebo pracovali. V závislosti na účelu je možné data exportovat v několika formátech a s použitím uživatelsky definovaných šablon. Exportovaná data je možno anonymizovat pro speciální případy.

Implementace se skládá ze serverové části s RESTovým API, SQLite databáze a klientské části. Vše je zabaleno do spustitelného balíčku, který funguje jako běžná desktopová aplikace. Implementovaný algoritmus netriviálním způsobem simuluje tvorbu pracovních záznamů podle předdefinovaných parametrů a zadaného chování zaměstnance. Vlastní řešení jsem zpracoval po analýze problému Knapsack, který nebylo možné použít kvůli nevhodnému hodnocení zisku.

Výsledkem této práce je kompletní analýza zahrnující sběr funkčních požadavků, případů užití a wireframy. Dále implementace podstatné části aplikace pro demonstraci simulačního algoritmu a následné testování UI a funkcí simulace.

Klíčová slova: počítačová aplikace; docházkový systém; zpracování docházky; export dat; anonymizace dat; problém batohu

The goal of this project is to analyse and implement attendance software by the assignment. Administrative workers use this application in a university office. It solves a problem of time-consuming attendance processing where the office is sorting the data by employees and project for the following processing. The goal of this application is to make the process and the export for third parties faster.

The application allows the user to manual input or batch import the employee's information, their current and past contracts. Based on a purpose, it is possible to export data in multiple different formats and with user-defined templates. For individual use cases, it is possible to anonymise the export.

The implementation consists of a server with RESTfull API, SQLite database and a client. Named are packed into an executable bundle which runs as a regular desktop application. The implemented algorithm simulates the creation of the work records in a nontrivial way using predefined parameters and given employee behaviour. After the Knapsack problem analysis, where I discovered it's not a suitable solution because of the items value system, I created the core algorithm.

The result of this work is complete analyses including functional requirements, use cases and user interface wireframes followed by implementation of the core part of the application to demonstrate the simulation and testing of a UI and the algorithm.

Keywords: computer application; attendance system; attendance processing; data export; data anonymization; knapsack problem

Title translation: System for attendance recording, generation and simulation

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

V bakalářské práci se zabývám analýzou a návrhem vhodného řešení pro záznam, generování a simulaci docházky.

Cílem je přinést řešení, které maximálně zjednoduší a zefektivní vytváření přehledů docházky dle různých kritérií, například z pohledu projektů nebo z pohledu jednotlivých zaměstnanců. Aktuální stav je absolutně bez dedikovaného systému a docházka se vytváří v tabulkovém procesoru na základě aktuálních požadavků.

Obsahem práce je zmapování aktuálního stavu, analýza domény, low fidelity návrh uživatelského rozhraní aplikace, implementace části aplikace a testování výsledného softwaru.

Cílem této práce není analyzovat a vytvořit software vyhovující zákoníku práce České republiky [1], popisy zaměstnanecké agendy se tomuto zákoníku [1] blíží, ale jen do takového detailu, který je potřebný pro vysvětlení problematiky a vytvoření softwaru fungujícího dle zadání. Práce se nezabývá optimalizací procesu záznamu docházky. Zaměřuje se především na generování jejích přehledů, bez ohledu jakým způsobem jsou data sbírána, či jsou-li manuálně zadávána.

Kapitola 2
Popis agendy spojené s vytvářením dokumentů docházky

2.1 Zaměstnanecké úvazky

Zaměstnanci mohou mít od jednoho do n úvazků. Přičemž pojem úvazek znamená, že fyzická osoba uzavřela pracovní smlouvu na dobu určitou nebo na dobu neurčitou. Pracovní náplň ve většině případů zahrnuje výzkumnou a pedagogickou činnost.

Zaměstnanecké úvazky se liší krom osobních údajů identifikujících danou osobu, ke které se úvazek vztahuje, také jeho velikostí, projekty, na kterých bude osoba vykonávat činnost, druhy činností a zaměstnavatelem.

2.1.1 Dělení podle velikosti úvazku

Pracovní doba je omezena shora zákoníkem práce České republiky [1] a mění se podle pracovní činnosti, její náročnosti a nebezpečnosti. Pro účely této práce ovšem používám zaměstnavatelem stanovený limit jako správný a hledím pouze na možné neshody s fyzikálními zákony. Nepřípustnými hodnotami jsou tedy časy, které nejsou fyzicky možné odpracovat, jelikož není tolik hodin v jednom dni, týdnu, měsíci či roku.

Dále také beru v potaz hranici spodní, která dle fyzikálních zákonů nemůže být záporná ani nulová.

Nulovou hodnotu zamítám pro její nejednoznačnost k záznamu docházky. Jelikož by mohla mít dvojí výklad. První možný by znamenal negativní informaci, že v daný okamžik, se osoba s nulovým záznamem na pracovišti nenacházela. To je ovšem zcela nerelevatní informace, jelikož z podstaty docházky se znamenávají především údaje o přítomnosti a zdůvodněné údaje o nepřítomnosti. Tedy záznam o nepřítomnosti bez bližší specifikace je nic neříkající.

Druhým výkladem by mohlo být, že osoba vykonala svojí činnost tak rychle, že odešla ve stejný okamžik kdy přišla. Tento výklad by vedl k řadě paradoxů a dalších nejasností, které jsou velmi vzdálené zadání a cílům této práce.

Příklad nulového nepřípustného záznamu může být příchod v 8:00 a odchod v 8:00 téhož dne.

Z maximální délky pracovní doby vychází velikost úvazku, kdy maximální možná hranice je plným úvazkem. Další velikosti úvazků lze poskytnout zaměstnavatelem v jakékoli výši podílu plného úvazku a jakéhokoliv čísla z oboru kladných reálných hodnot bez nuly.

2.1.2 Dělení podle projektů

Každý zaměstnanec pracuje alespoň na jednom projektu. A to i v případě, že vykonává obecnou činnost, kterou nelze jednoznačně přiřadit jednomu projektu. V takových případech je vytvořen obecný projekt pro tento druh obecné činnosti.

Typickým příkladem takovýchto činností je pedagogická a administrativní činnost.

2.1.3 Dělení podle činnosti

Pracovní činnost, zkráceně „činnost“, která je uvedena ve smlouvě mezi zaměstnancem a zaměstnavatelem přímo určuje, o jaký druh zaměstnance se jedná a může specifikovat, na kterých projektech danou činnost může vykonávat.

Pedagogická činnost implikuje druh akademického zaměstnance. Činnost administrativní implikuje druh zaměstnance administrativního. A činnost technická implikuje druh zaměstnance technického. Další druhy pro účely použití analyzované aplikace nejsou potřeba.

2.1.4 Dělení podle druhu

Toto dělení je shodné s dělením podle činnosti. Druh zaměstnance určuje profil činnosti, kterou bude zaměstnanec vykonávat, neurčuje však konkrétní projekty na rozdíl od činnosti.

Praktickým příkladem dělení podle druhu je například druh administrativní pracovník. Přiřazením tohoto druhu k zaměstnanci umožňuje obecně specifikovat, na kterých projektech bude zaměstnanec moci pracovat. V tomto případě tedy pouze na projektech, kde je vyvíjena administrativní činnost. V tom je zásadní rozdíl při porovnání s dělením podle činnosti, kde konkrétně specifikujeme činnosti v projektech.

2.1.5 Dělení podle zaměstnavatelů

Pro úplnost popisu zaměstnaneckých úvazků uvádím i dělení podle zaměstnavatele. Ovšem pro účely této práce se nejedná o významný atribut pracovní smlouvy, jelikož výsledná aplikace je koncipovaná pro jednoho zaměstnavatele.

Zároveň ale myslím na rozšiřitelnost a přehlednost, a proto tato dělení zahrnuji i v následujících částech práce.

2.2 Vymezení systému docházky a jeho účelů

Systém docházky, zkráceně „docházka“, je způsob vyhodnocování a sledování pracovní činnosti včetně pracovních cest, všech druhů dovolených a dalších nepřítomností. Přímo umožňuje získat informace o skutečné pracovní době jednotlivých pracovníků. Nepřímo s použitím kontextu docházky ostatních pracovníků nám umožňuje sledovat skutečnou pracovní dobu skupin i celé společnosti.

Získané informace o odpracovaných hodinách a dovolených jsou typicky použity pro účely kontroly nebo výpočtu mzdy. Pro účely kontroly se může jednat o interní kontrolu vybraných zaměstnanců nadřízenými nebo kontrolu od externího subjektu, který je smluvně spojen s prací, kterou zaměstnanci vykonávají a ze smlouvy vyplývá nárok na kontrolu odpracovaných hodin, čerpání dovolené a dalších informací z docházky. V tomto případě je důležité věnovat pozornost informacím, které se na přehledu docházky zobrazují, aby nedocházelo k neoprávněnému informování třetí strany o práci na jiných projektech, případně narušení soukromí zaměstnanců. Tedy informace o pracovní činnosti nesouvisející se stranou, která si přehled docházky vyžádala, nemůžou být této straně přípustné a záznamy musí být generalizované.

Dále tyto informace mohou být použity pro přehledy výkonu pracovníků nebo přehledy průběhu práce na projektech, oboje v porovnání s dalšími daty, například odhady pracnosti v člověkohodinách.

2.2.1 Aktivity zahrnuté v docházce

Aktivity nebo-li pracovní činnosti zahrnují vše, co zaměstnanec dělá jak v pracovní době, tak mimo ni, ovšem se zachováním jeho soukromí a dobrých mravů. Tedy záznamy v rámci pracovní doby obsahují větší míru detailu v porovnání se záznamy mimo pracovní dobu.

Záznamy mimo pracovní dobu a náplň práce rozlišují mezi volným časem, dovolenou, placeným volnem, víkendy, státními svátky, přestávkami na jídlo a oddechem. Záznamy v pracovní době nebo-li pracovní záznamy zahrnují všechny činnosti vykonávané podle pracovní smlouvy. Pro účely zaznamenávání docházky rozlišuji jen mezi omezeným množstvím pracovních činnosti a to mezi takovými, jejichž rozlišení je potřebné k určení přítomnosti zaměstnance na pracovišti či nikoliv. Jedná se o následující aktivity, obecná práce na projektu, projektový meeting, pracovní cesta v rámci práce na projektu. U pedagogických pracovníků navíc rozlišuji vedení přednášek, cvičení, konzultacích a obecnou pedagogickou činnost spojenou s vyučovanými předměty.

2.3 Procesní pohled na záznam docházky

Podle zadání této práce jsem systém docházky rozdělil do třech hlavních procesů, které zachycují celý průběh tohoto systému od začátku do konce, tedy od záznamu docházky po vytvoření přehledu pro kontrolu. Prvním procesem na časové ose je Záznam docházky, kdy zaměstnanci při své pracovní činnosti vytvářejí a zaznamenávají data pro docházku. Zpracování docházky je dalším procesem v pořadí, kdy se data předají ke zpracování administrativním pracovníkům. Od tohoto bodu je problém práce s docházkou součástí mé práce. Posledním krokem je Získávání informací docházky, kdy administrativní pracovníci předají přehledy docházky další straně, nadřízeným nebo jiné společnosti.

2.3.1 Záznam

Jelikož tento proces předchází části, kterou se zabývá tato práce, jen krátce naznačím, co se uvnitř odehrává a jaké jsou možnosti spojení s navazujícím procesem.

Záznam dat docházky může být generovaný automaticky pomocí přístupových karet a systémů pro záznam dovolených a pracovních cest. Nebo může být zcela manuální, kdy si zaměstnanci sami vedou záznamy o příchodech, odchodech, pauzách, dovolených a dalších.

Jakkoliv získaná data jsou předána aktivně zaměstnanci nebo sběrem administrativního pracovníka do administrativního oddělení společnosti, kde budou tato data zpracována pro další použití. Předáním dat končí proces záznamu dat docházky.

2.3.2 Zpracování

Zpracování dat je první z procesů systému docházky, kterým se tato práce zabývá. Navazuje na předchozí záznam docházky a začíná v okamžiku příchodu dat na administrativní oddělení, kde jsou data přiřazena k jednotlivým pracovníkům a projektům. Dále jsou data roztříděna podle druhu na záznamy o práci, pracovní cestě, druhu dovolené, přestávky, či pracovní činnosti. Proces končí zpracováním všech dat na vstupu tohoto procesu.

2.3.3 Získávání informací

Druhým procesem, na který se v této práci zaměřím, a posledním z hlavních procesů systému docházky je Získávání informací. Zpracovaná data z předchozího procesu získají

informační hodnotu v závislosti na vybraném kontextu. Tím může být časové období, skupina pracovníků, či skupina záznamů podle příslušnosti k projektu. Tento kontext je vymezen požadavkem na administrativní oddělení, které poskytuje přehledy docházky například pro účely kontroly. Získávání informací končí předáním požadovaného přehledu žadateli.

2.4 Přínos použití softwaru

Agenda docházky, viz 2, a její procesy, viz 2.3, jako takové nejsou nijak závislé na technologiích a celý systém docházky může být provozován ručně. Data mohou být zapisována do tabulek a sešitů v papírové podobě. To bude velmi časově náročné a zároveň velmi náchylné na zanesení chyby. Pokud se přesuneme o krok vpřed s použitím moderních technologií můžeme použít některý z dnes dostupných tabulkových procesorů. Toto řešení je však stále velmi časově náročné a náchylné na chybu. Přesto již přináší výhody moderního světa v podobě jednoduchého zálohování a korekcí oproti fyzickým tabulkám. Posledním krokem v současné době je použití specializovaného softwaru, jak uvádí konceptuální model pro automatický docházkový systém s použitím rozpoznávání obličeje [2].

Specializované technologie přinášejí větší spolehlivost, úsporu času a možnosti dalšího zpracování dat. Generování přehledů může být softwarem provedeno během několika sekund a to i v případě zpracovávání několika zaměstnanců najednou. Oproti tomu získávání informací z fyzické či elektronické tabulky pro několik zaměstnanců najednou může zabrat až hodiny práce. A výsledné přehledy jsou vystaveny velkému riziku výskytu chyb.

Kapitola 3
Analýza požadavků systému

3.1 Požadavky

V následujících odstavcích jsem zaznamenal všechny funkční a kvalitativní požadavky na software, který bude sloužit k podpoře procesů popsaných v předchozí kapitole této práce. Sběr požadavků probíhal především diskuzí se zadavatelem práce, potencionálními uživateli softwaru a zaměstnanci, jejichž činnosti bude v tomto kontextu zaznamenávána.

3.1.1 Funkční požadavky

Následující funkční požadavky jsou označené zkratkou „FR X.Y“, kde písmena „FR“ jsou vzata z anglického názvu pro funkční byznys požadavky, functional business requirements, a „X.Y“ jsou nahrazena číslem kapitoly a pořadovým číslem požadavku.

Jelikož jediným uživatelem systému je administrativní pracovník, již tuto informaci neopakuji v jednotlivých specifikacích funkčních požadavků pod tímto textem.

Následující požadavky se týkají práce se záznamy docházky.

FR 3.1. Systém bude umožňovat zadávat údaje o docházce. Konkrétně bude umožňovat zadávat následující údaje:

- Datum a čas začátku záznamu – typicky se bude jednat o příchod na pracoviště, zahájení dovolené nebo přestávky.
- Datum a čas konce záznamu – jedná se o konce aktivit popsaných výše.
- Projekt – jehož se práce týkala.
- Typ záznamu – určuje, zda se čas počítá do pracovní doby zaměstnance, nebo se počítá do času dovolené zaměstnance, nebo některý z dalších druhů viz analýza problému v předcházející části tohoto dokumentu.

FR 3.2. Systém bude umožňovat změnu záznamů zadaných podle požadavku 3.1.

FR 3.3. Systém bude umožňovat odstranění záznamu docházky zadaného podle požadavku 3.1.

Následující požadavky se týkají práce se zaměstnanci.

FR 3.4. Systém bude umožňovat přidání, změnu a odstranění zaměstnance s následujícími atributy:

- Celé jméno
- Akademické tituly
- Datum narození
- Celkový úvazek – vyjádřen v zleva polootevřeném intervalu od 0 do 1.
- Specifikace úvazku – bude možné vybrat, na jakých projektech se zaměstnanec podílí v jakém časovém období na jak velký úvazek.

FR 3.5. Systém bude umožňovat zobrazení a úpravu přehledu zaměstnance s informacemi z požadavku 3.4 a následujícími:

- Přehled projektů – kterých se zaměstnanec účastní nebo účastnil.
- Přehled docházky – který zobrazí časy a souvisejících činnosti. Například příchod, odchod nebo služební cesta.

  Výše zmíněný přehled docházky bude umožňovat následující filtrování:

  - Výběr projektu – kterého se docházka týká.
  - Výběr časového období – ve volitelném rozmezí od do.

FR 3.6. Systém bude umožňovat zobrazení přehledu všech zaměstnanců, který lze filtrovat podle následujících parametrů. Rok a měsíc, kdy byla osoba alespoň po jeden kalendářní den zaměstnaná.

Následující požadavky se týkají práce s přehledy projektů a pohledů na jednotlivé projekty.

FR 3.7. Systém bude umožňovat přidání, úpravu a odstranění projektu s následujícími atributy:

- Název
- Datum zahájení a ukončení

Zaměstnanec se bude moci účastnit několikrát po sobě s různou velikostí úvazku.

FR 3.8. Systém bude umožňovat zobrazení detailu projektu s následujícími atributy:

- Časové období – datum zahájení a ukončení projektu
- Seznam všech zaměstnanců účastnících se projektu

  U zaměstnanců budou zobrazeny následující položky:

  - Jméno
  - Popis práce na projektu
  - Velikost úvazku
  - Odpracované hodiny za zobrazované období
  - Zahájení úvazku
  - Konec úvazku

FR 3.9. Systém bude umožňovat zobrazení přehledu všech projektů, který lze filtrovat podle následujících parametrů. Rok a měsíc, ve kterém projekt běžel alespoň v jednom kalendářním dni.

Následující požadavky se týkají práce se šablonami pro generování přehledů.

FR 3.10. Systém bude umožňovat zobrazení přehledu všech šablon pro generování přehledů.

FR 3.11. Systém bude umožňovat odstranění šablony ze systému.

FR 3.12. Systém bude umožňovat vytvoření šablony pro generování všech druhů přehledů. Tvorba šablon bude probíhat pomocí skládání bloků s přednastavenou funkcionalitou a vzhledem, viz tento seznam:

- Karta zaměstnance
- Karta projektu
- Týdenní graf
- Roční tabulka docházky
- List záznamů docházky
- A další...

Následující požadavky se týkají exportu dat neboli generování přehledů.

FR 3.13. Systém bude umožňovat generování přehledů pro jednotlivé zaměstnance se specifikací následujících údajů:

- Projekty – které se mají zobrazit v přehledu. Vybírá se pouze z těch, kterých se zaměstnanec účastnil v daném časovém období.
- Časové období – s možností výběru počátečního a koncového dne. Periody, kdy zaměstnanec pracoval na jiných než vybraných projektech, nebo čerpal dovolenou z jiného projektu, budou anonymizována.

Exportované přehledy bude možné získat v následujících formátech:

- CSV
- PDF

FR 3.14. Systém bude umožňovat generování přehledů pro jednotlivé projekty se specifikací následujících údajů:

- Časové období – s možností výběru počátečního a koncového dne.

Exportované přehledy bude možné získat v následujících formátech:

- CSV
- PDF

Následující požadavky se týkají hromadného importu dat.

FR 3.15. Systém bude umožňovat import zaměstnanců a jejich úvazků z CSV souborů.

FR 3.16. Systém bude umožňovat import projektů a jejich detailů z CSV souborů. S projektem bude možné importovat následující atributy:

- Název projektu
- Datum zahájení ukončení projektu

FR 3.17. Systém bude umožňovat import docházky zaměstnanců z CSV souborů. Záznam docházky pro import se skládá z následujících údajů:

- Začátek a konec záznamu s údajem o dni a času
- Typ záznamu. Například práce na projektu, čerpání dovolené a podobně.
- Identifikátor projektu. Může být například jméno projektu.

Následující požadavky se týkají provádění simulace docházky.

FR 3.18. Systém bude umožňovat simulaci práce na projektu u jednotlivých zaměstnanců. Simulace se bude řídit následujícími parametry:

- Zůstanou zachována již existující data
- Simulace má datum zahájení a datum konce
- Zaměstnanci nepracují o státních svátcích
- Zaměstnanec má klouzavý začátek pracovního dne
- Pracovní doba je klouzavá
- Dovolená se čerpá rovnoměrně ze všech úvazků

Následující požadavky jsou výše nezařazené a jedná se o doménově nezávislé požadavky aplikovatelné obecně na informační systémy.

FR 3.19. Systém bude umožňovat přihlášení uživatelů pomocí uživatelského jména a hesla. Přístup do systému bude jeho uživatelům zajistěn administrátorem.

3.1.2 Kvalitativní požadavky

Následující funkční požadavky jsou označené zkratkou „NFR X.Y“, kde písmena „NFR“ jsou vzata z anglického názvu pro kvalitativní byznys požadavky, non-functional system requirements, a „X.Y“ jsou nahrazena číslem kapitoly a pořadovým číslem vzhledem ke všem požadavkům. To znamená i funkčním.

NFR 3.20. Systém bude ve své implementaci využívat software třetích stran, jehož licence umožňuje bezplatné využití pro komerční účely.

NFR 3.21. Přehledy generované do formátu PDF a CSV v objemu do 100 budou dostupné ke stažení uživateli do 1 minuty od zahájení generování. Každých dalších 100 položek zvyšuje maximální přípustný limit o 0,5 minuty.

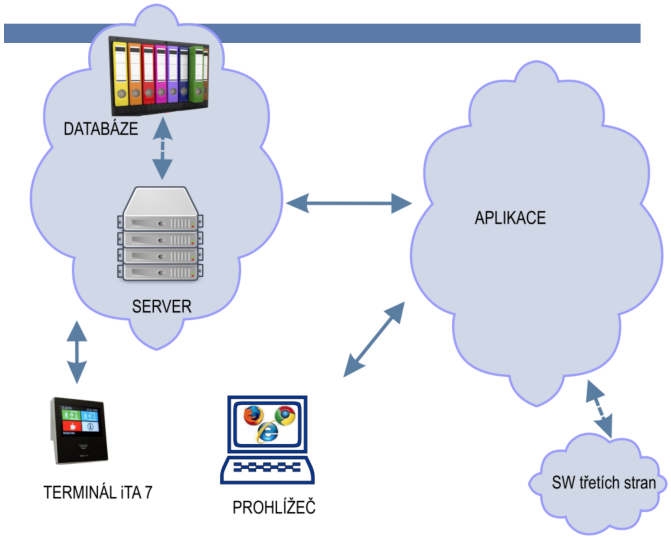
Kapitola 4
Existující řešení

Podle zadání této práce a získaných funkčních požadavků, viz kapitola číslo 3 Analýza požadavků, jsem zjistil, že existují řešení, která jsou obecně nazývána jako docházkové systémy a která více či méně naplňují požadavky mnou navrhovaného a vyvíjeného systému.

Každé řešení, se kterým jsem se při tvorbě této práce setkal, bylo spojené s hardwarovým řešením zajišťujícím sběr docházkových dat. Což pokud se vrátím k popisu řešeného problému, viz 2.3, znamená zásah všech zmíněných procesů. Nicméně tato práce se zaměřuje pouze na poslední dva, Zpracování a Získávání informací. Tedy i v následujícím popisu existujících řešení se zaměřuji především na softwarovou stránku docházkových systémů a na jejich naplnění získaných požadavků.

4.1 iTA

Docházkový systém iTA[1] je vyvíjen společností ELEKON s. r. o.[2], která je členem mezinárodního holdingu MOSER-BAER AG se sídlem ve Švýcarsku. Tento systém je velmi úzce spojen se širokým sortimentem docházkového hardwaru, který společnost taktéž vyvíjí. Systém iTA je dle dostupných informací možné používat i bez dalšího docházkového hardwaru, ovšem společnost sama to nikde nedoporučuje. Jako ideální součást instalace systému iTA je zmíněn alespoň jeden docházkový terminál iTA 7 připojený k internetu. [3]



Obrázek 4.1. Diagram nasazení systému iTA. Zdroj [3].

[1] Produktový web systému iTA `https://firemnidochazka.cz/`. Aktuální k 29. 12. 2019.
[2] Oficiální webové stránky společnosti ELEKON s. r. o. `https://mobatime.cz/`. Aktuální k 29. 12. 2019.

Hlavní předností systému iTA je webová aplikace, díky které se zákazníci nemusejí starat o provoz samotného softwaru. Nemusejí tedy řešit aktualizace ani bezpečnost z pohledu technologie. Aplikace umožňuje připojení systémů třetích stran, například od společnosti SAP SE[1]. Systém splňuje požadavky obecného nařízení pro ochranu osobních údajů, GDPR[2].

Aplikace umožňuje provádět automatický záznam docházky z docházkových terminálů nebo přidávat záznamy ručně přes webové rozhraní. Takzvané „zařazení zaměstnance“ umožňuje změnu nastavení pravidel docházky na základě pracovní pozice a místa výkonu. Tato funkce naplňuje částečně funkční požadavek 3.18, který taktéž vyžaduje přiřazení metadat o pravidlech docházky k jednotlivým zaměstnancům. Ovšem systém iTA tato data využívá k hodnocení docházky a stanovení času, kdy by měl zaměstnanec pracovat.

Pro záznam aktivit mimo standardní pracovní dobu, například přesčas, nemoc, dovolená, služební cesta, využívá systém iTA časové účty, kterým lze definovat krom jejich názvu dodatečné vlastnosti jako limit čerpání nebo akce pro zpracování zůstatku na účtu, například převod do dalšího měsíce. Pohyby na účtech lze dále analyzovat a výstupy přidávat do přehledů docházky.

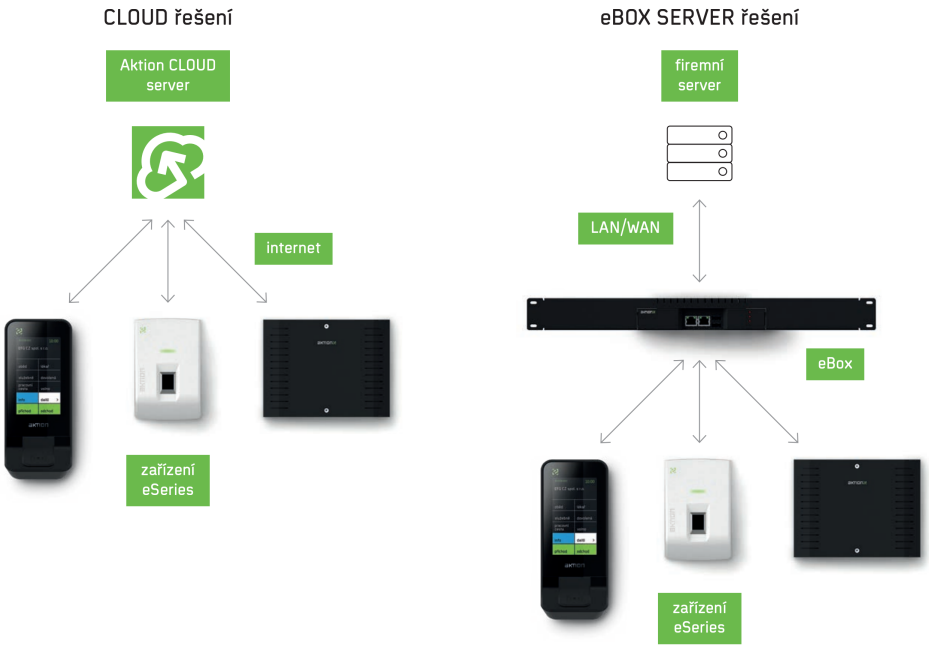
Systém dále umožňuje nastavení takzvaných „sestav“ pro tisk, email a další exporty. V tomto případě sestavu můžu přirovnat k šablonám z funkčního požadavku 3.12. Export sestav a přehledů je umožněn ve formátech PDF a CSV stejně jako u funkčních požadavků 3.13 a 3.14.

Systém iTA speciálně neřeší zařazení zaměstnanců do projektů, ovšem jeho konfigurace na míru umožňuje splnění tohoto požadavku nepřímo. Například využitím speciálních časových účtu.

4.2 Aktion

Docházkový systém Aktion[3] je vyvíjen společností EFG CZ s. r. o.[4], která svůj docházkový systém dodává například společnostem Alza.cz a.s., ŠKODA AUTO a.s. nebo z bankovního sektoru Citybank. Systém Aktikon je možné využívat s použitím aplikací pro mobilní telefony, tablety a počítače nebo s využitím docházkových snímačů řady eSeries verze provedení eSmartReader. [4]

[1] Web společnosti SAP SE `https://www.sap.com/`. Aktuální k 1. 1. 2020.
[2] Oficiální znění GDPR `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679`. Aktuální k 1. 1. 2020.
[3] Produktové weby systému Aktion `https://www.aktion.cz/` a `https://www.dochazkaonline.cz/`. Aktuální k 1. 1. 2020.
[4] Oficiální webové stránky společnosti EFG CZ s. r. o. `https://www.efg.cz/`. Aktuální k 1. 1. 2020.



Obrázek 4.2. Diagram nasazení systému Aktion. Zdroj [4].

Oproti konkurenci systém Aktion umožňuje zaznamenávat docházku krom webového rozhraní a docházkového terminálu také pomocí nativní aplikace pro mobilní telefony a tablety. Funkce docházkový předpis umožňuje nastavit pracovní dobu zaměstnanců, očekávané doby příchodů a odchodů a další pravidla docházky. Čehož se následně využívá při zpracování dat v době, kdy ještě není v daný den známý čas odchodu zaměstnance. Toto částečně naplňuje funkční požadavek 3.18, který navíc vyžaduje simulaci klouzavé pracovní doby, ovšem to systém Aktion nepodporuje. Ten pouze použije hodnotu podle daného pravidla docházky.[5]

Nasazení systému Aktion je možné dvěma způsoby. Prvním a také nejsnadnějším způsobem z pohledu instalace hardwaru je cloudové řešení, kdy jsou všechna data zpracována na serverech poskytovatele řešení. Druhým, náročnějším způsobem, jelikož je nutná instalace a zabezpečení vlastního serveru, je použití se zařízením eBox, viz katalogový list systému Aktion [4], které umožňuje připojení nabízeného docházkového hardwaru k zmíněnému lokálnímu serveru.

Systém umožňuje uživatelům používat přednastavené sestavy, které umožňují měnit obsah zobrazovaných a následně exportovaných dat. V standardním režimu aplikace jsou dostupné například měsíční přehledy, pracovní cesty, přítomnost a další, viz dokumentace systému Aktion [5]. Ovšem není uživatelsky možné přidat si vlastní sestavu, což není v souladu s požadavkem 3.12, který vyžaduje uživatelsky nastavovatelné šablony. Přehledy docházky lze exportovat do několika formátů, například PDF, XLS, XLSX, RTF, CSV a dalších, což splňuje požadavek 3.13 a 3.14 v oblasti nabízených formátů exportovaných dat.

Podobně jako u konkurenčních produktů systém Aktion neřeší zařazení zaměstnanců do projektů. Nicméně nabízí široké možnosti filtrování, které tuto funkcionalitu mohou částečně nahradit.

4.3 Závěr

Žádné ze zmíněných řešení popsaných výše, viz tato kapitola 4, nesplňuje funkční požadavky specifikované v kapitole 3. Proto v následujících kapitolách pokračuji v návrhu a implementaci požadovaného systému.

Nejzásadnějším problémem s analyzovanými řešeními byla absence správy projektů a jejich přiřazení k zaměstnancům. Dále také nebyl splněn požadavek na simulaci docházky a to úplnou absencí u systému iTA, viz 4.1, nebo nesplněním specifikace u systému Aktion, viz 4.2. Posledním zásadním problémem je anonymizace dat, kterou systémy splňují jen částečně, pokud použijeme jejich rozsáhlé možnosti filtrování.

Mimo těchto zřejmých nesplnění funkčních požadavků je také problém v zaměření zkoumaných systémů na sběr dat z hardwarových zařízení, která tyto společnosti vyrábí a se softwarem dodávají. Použití bez hardwaru je možné, ovšem manuální zadávání dat je vždy považováno jako druhořadý vstup, tedy není optimalizováno pro dávkové importy a velké množství dat.

Kapitola 5
Návrh řešení

V této kapitole popisuji návrh doménového modelu a analýzu případů použití aplikace, podle výše získaných funkčních a kvalitativních požadavkům, viz 3.

5.1 Analýza případů použití

V následujícím textu popisuji případy použití analyzované aplikace podle specifikace UML [6].

Jelikož je aplikace určena pouze pro administrativní pracovníky a vzhledem k její velikosti není vyžadována žádná správa práv nebo komplikované technické nastavení, jsou tito pracovníci jedinými aktéry v systému a budou se k nim vztahovat všechny případy použití popisované v následujícím textu.

Následující případy použití jsou označeny zkratkou „UC X.Y“, kde písmena „UC“ jsou vzata z anglického názvu pro případ použití, use case, a „X.Y“ jsou nahrazena číslem kapitoly a pořadovým číslem případu použití.

Každý z případů použití začíná iniciativou ze strany uživatele, proto ji ve scénářích u následujících případů použití nebudu uvádět.

Přestože scénáře pro triviální případy použití nejsou potřeba pro jejich objasnění, uvádím je v následujícím textu pro zachování konzistence, při odkazování na příslušné návrhy obrazovek.

UC 5.1. Přidat údaje o docházce zaměstnance. Mapuje požadavek 3.1.
Scénář:

1) Systém zobrazí formulář pro přidání záznamu docházky. Wireframe G.3.
2) Uživatel vyplní nebo upraví požadované údaje a potvrdí je.
3) Systém provede kontrolu zadaných údajů.

   Když budou údaje v pořádku, Systém údaje uloží a zobrazí obrazovku, na které se nacházel před *1. bodem* tohoto scénáře.

   Jinak Systém zobrazí vyplněné údaje nezměněné ve stejném formuláři a informuje uživatele o údajích, které jsou v nepořádku, a o datech, které zadané údaje porušily. A vrátí se na *2. bod* tohoto scénáře.

Kontrola údajů se skládá ze dvou částí, formální a faktické. První z uvedených se zabývá formátem zadaných dat a kontroluje, zda je možné takto uvedená data systémem dále zpracovat. Tato kontrola probíhá přímo v uživatelském rozhraní.

Faktická kontrola přidá zadané údaje k již existujícím a kontroluje, zda nebyly překročeny limity vyplývající z velikosti úvazku a smlouvy zaměstnance. Faktická kontrola může znamenat delší dovolenou, než na kterou má zaměstnanec nárok dle velikosti jeho úvazku a podobné případy.

UC 5.2. Upravit údaje o docházce zaměstnance. Mapuje požadavek 3.2.
Scénář:

1) Systém zobrazí formulář pro úpravu záznamu docházky s daty vybraného uživatelem vybraného záznamu. Wireframe G.4.
2) Uživatel vyplní nebo upraví požadované údaje a potvrdí je.
3) Systém provede kontrolu zadaných údajů stejně jako v případě scénáře 5.1.

UC 5.3. Zobrazit údaje o docházce zaměstnance. Mapuje požadavek 3.5.
Scénář:

1) Systém zobrazí obrazovku záznamů o docházce vybraného zaměstnance. Wireframe G.2.

UC 5.4. Zobrazit údaje o projektech, na kterých zaměstnanec pracuje. Mapuje požadavek 3.5.
Scénář:

1) Systém zobrazí obrazovku projektů vybraného zaměstnance. Wireframe G.5.

UC 5.5. Přidat projekt k zaměstnanci. Mapuje požadavek 3.4.
Scénář:

1) Systém zobrazí formulář pro přidání projektu k zaměstnance. Wireframe G.6.
2) Uživatel vyplní nebo upraví požadované údaje a potvrdí je.
3) Systém provede kontrolu zadaných údajů.
   Když budou údaje v pořádku, Systém údaje uloží a zobrazí obrazovku, na které se nacházel před *1. bodem* tohoto scénáře.
   Jinak Systém zobrazí vyplněné údaje nezměněné ve stejném formuláři a informuje uživatele o údajích, které jsou v nepořádku, a o datech, které zadané údaje porušily. A vrátí se na *2. bod* tohoto scénáře.

Kontrola údajů se skládá ze dvou částí, formální a faktické. První z uvedených se zabývá formátem zadaných dat a kontroluje, zda je možné takto uvedená data systémem dále zpracovat. Tato kontrola probíhá přímo v uživatelském rozhraní.

Faktická kontrola zařadí nový úvazek mezi stávající a zkontroluje, že všechny úvazky dohromady nepřekračují velikost celkového úvazku dle smlouvy zaměstnance na dané období.

Dále systém provede kontrolu časového úseku vůči smlouvě na daný časový úsek.

UC 5.6. Přidat zaměstnance do projektu. Mapuje požadavek 3.7.
Scénář:

1) Systém zobrazí formulář pro přidání zaměstnance do projektu. Wireframe G.7.
2) Uživatel vyplní nebo upraví požadované údaje a potvrdí je.
3) Systém provede kontrolu zadaných údajů stejně jako v případě scénáře 5.5.

UC 5.7. Zobrazit přehled zaměstnanců. Mapuje požadavek 3.6.
Scénář:

1) Systém zobrazí obrazovku s přehledem všech zaměstnanců. Wireframe G.8.

UC 5.8. Zobrazit přehled projektů. Mapuje požadavek 3.9.
Scénář:

1) Systém zobrazí obrazovku s přehledem všech projektů. Wireframe G.9.

UC 5.9. Zobrazit přehled šablon. Mapuje požadavek 3.10.
Scénář:

1) Systém zobrazí obrazovku s přehledem všech šablon. Wireframe G.10.

UC 5.10. Generovat docházku zaměstnanců. Mapuje požadavek 3.13.
Scénář:

1) Systém zobrazí obrazovku s přehledem všech zaměstnanců. Wireframe G.8.
2) Uživatel vybere takové zobrazované období, aby mohl vybrat všechny zaměstnance, v rozsahu 1 až n, kde n je počet zaměstnanců, pro které chce generovat docházku. Poté uživatel vybere zaměstnance.
   Uživatel vybere volby generování. Zda se generují nulové přehledy a zda se do generovaných přehledů zahrnuje simulace. Wireframe G.8.
3) Systém zobrazí výběr výstupních formátů generovaných dokumentů. Wireframe G.8.
4) Uživatel vybere formát dokumentu a potvrdí generování.
5) Systém provede kontrolu vstupní dat pro generování.
   Když nebude vybrána volba generovat nulové přehledy a systém nalezne nulový přehled.

   1) Systém zobrazí varování před nulovým přehledem.
   2) Uživatel potvrdí toto varování.
   3) Systém se vrátí na *1. bod* tohoto scénáře s aktuálním výběrem zaměstnanců (aktuálními daty).

   Jinak Systém zobrazí obrazovku s informací o stavu generování dokumentů. Wireframe G.11 A po vygenerování zobrazí seznam dokumentů ke stažení. Wireframe G.12
6) Uživatel si uloží vygenerované dokumenty.

Kontrola vstupních dat, konkrétně kontrola nulových přehledů, se provádí, protože pokud se najde zaměstnanec, který ve vybraném časovém rozmezí nebyl součástí projektu nebo který nemá v daném období zaznamenanou v pracovní době docházku, jedná je pravděpodobně o chybu, kterou je nutné před generováním opravit.

Případně pokud si je této skutečnosti uživatel vědom a přesto chce přehledy vygenerovat, může použít volbu generování nulových přehledů, poté systém tuto validaci přeskočí.

UC 5.11. Generovat docházku z projektu. Mapuje požadavek 3.14.
Scénář:

1) Systém zobrazí přehled všech zaměstnanců účastnících se projektu. Wireframe G.13. Dále pokračuje shodně s případem použití 5.10.

Kontrola vstupních dat probíhá stejně jako u případu použití 5.10.

UC 5.12. Generovat docházku zaměstnance. Mapuje požadavek 3.13.
Scénář:

1) Systém zobrazí přehledem docházky zaměstnance. Wireframe G.2.
2) Uživatel vybere zobrazované období, za které chce vygenerovat přehled.
3) Systém zobrazí výběr výstupních formátů generovaných dokumentů. Wireframe G.2.
4) Uživatel vybere formát dokumentu a potvrdí generování.
5) Systém provede kontrolu vstupní dat pro generování.
   Když nebude vybrána volba generovat nulové přehledy a systém nalezne nulový přehled.

   1) Systém zobrazí varování před nulovým přehledem.
   2) Uživatel potvrdí toto varování.
   3) Systém se vrátí na *1. bod*.

   Jinak Systém zobrazí obrazovku s informací o stavu generování dokumentů. Stejné modální okno jako wireframe G.11 A po vygenerování zobrazí dokument ke stažení. Obdobně jako u wireframe G.12, ale s jedním dokumentem.
6) Uživatel si uloží vygenerované dokumenty.

Kontrola vstupních dat probíhá stejně jako u případu použití 5.10.

UC 5.13. Vytvořit šablonu pro generování docházky. Mapuje požadavek 3.12.
Scénář:

1) Systém zobrazí editor šablon s načtenou vybranou šablonou. Wireframe G.14.
2) Uživatel přidá nabízené komponenty do šablony a vyplní požadované údaje a potvrdí je. Wireframe G.14.
3) Systém provede kontrolu zadaných údajů a uloží šablonu.

Kontrola požadovaných údajů spočívá v kontrole maximální délky názvu šablony a povolených znaků.

UC 5.14. Zobrazit detail projektu projektu. Mapuje požadavek 3.8.
Scénář:

1) Systém zobrazí obrazovku s detailem projektu. Wireframe G.13.

UC 5.15. Přidat zaměstnance. Mapuje požadavek 3.4.
Scénář:

1) Systém zobrazí obrazovku s formulářem pro přidání nového zaměstnance. Wireframe G.13.
2) Uživatel vyplní všechny povinné údaje a další volitelně.
3) Systém provede kontrolu uvedených údajů.
   Když Systém nalezne chybu zobrazí pro každou chybu špatnou hodnotu a její limit nebo správný tvar a přejde na *1. bod* tohoto scénáře.
   Jinak Systém uloží data.

Systém kontroluje zda součet všech úvazku zaměstnance, které probíhají současně, nepřekračuje jeden celý úvazek. Pokud ano, systém varuje uživatele o této skutečnosti, ale stále umožní data uložit.

Podobně jako celé úvazky systém dále kontroluje, že suma velikosti dílčích úvazků, které probíhají současně, nepřekračuje velikost úvazku, ke kterému se pojí.

Pokud existuje úvazek a dílčí úvazek a uživatel zadá upřesňující data k simulaci, systém jej upozorní, pokud počet pracovních hodin za vybrané dny v týdnu je menší než počet pracovních hodin za týden se zadanou velikostí úvazků.

UC 5.16. Přidat projekt. Mapuje požadavek 3.7.
Scénář:

1) Systém zobrazí obrazovku s formulářem pro přidání nového projektu a pro každého zaměstnance vypíše jméno a zobrazí, zda byl nalezen konflik v období, kdy se zaměstnanec bude účastnit projektu. Wireframe G.16.
2) Uživatel vyplní všechny požadované údaje a potvrdí je.
3) Systém provede kontrolu konfliktů.
   Když bude nalezen konflik.
   1) Systém varuje uživatele o vytváření projektu s konfliktem a informuje uživatele o tom, u kterého zaměstnance nastal konflikt.
      Když uživatel potvrdí vytvoření s konfliktem.
      1) Systém vytvoří nový projekt.

      Jinak uživatel zamítne vytvoření s konfliktem.
      1) Systém se vrátí na *1. bod* tohoto scénáře.

   Jinak Systém vytvoří nový projekt.

Konflikt spočívá v pokusu o zapsání většího pracovního hodinového vytížení, než, které má zaměstnanec zapsané v systému, respektive ve smlouvě. Konflikt uživatel může vyřešit při tvorbě nového projektu snížením úvazku na právě vytvářeném projektu, pokud se například jednalo o překlep, nebo lze projekt vytvořit s konfliktem a následně situaci vyřešit v detailu uživatele.

Po dobu konfliktu nelze generovat přehledy zaměstnanců ani projektů.

UC 5.17. Importovat zaměstnance. Mapuje požadavek 3.15.
Scénář:

1) Systém zobrazí dialogové okno operačního systému pro výběr souboru.
2) Uživatel potvrdí výběr souboru.
3) Systém informuje uživatele o průběhu importu. Systém informuje uživatele o následujích událostech:
   - Neimportované položky z důvodu duplicity.
   - Neimportované položky z důvodu chybějících údajů, špatných typů u jedno a více atributů a dalších neočekávaných chyb.
   - Celkový počet importovaných položek.
   - Celkový počet neimportovaných položek.
   - Importované položky.

Informování o importovaných položkách může být provedeno například seznamem položek nebo jako zvýraznění či jiné označení nově importovaných položek v seznamu či přehledu, například přehled všech zaměstnanců, případ použití 5.7 nebo přehled zaměstnanců pracujících na projektu, případ použití 5.14.

UC 5.18. Importovat projekt. Mapuje požadavek 3.16.
Importování projektu probíhá shodně jako případ použití 5.17.

UC 5.19. Importovat docházku. Mapuje požadavek 3.17.
Importování docházky probíhá shodně jako případ použití 5.17.

UC 5.20. Simulovat docházku. Mapuje požadavek 3.18.
Scénář:

1) Systém zobrazí parametry simulace.
2) Uživatel upraví parametry a potvrdí je.
3) Systém doplní nezměněné či chybějící údaje výchozími hodnotami.
4) Uživatel spustí simulaci.
5) Systém informuje o průběhu simulace.

Simulace je typicky spustitelná pro jednotlivé zaměstnance nebo pro jednotlivé projekty.

5.2 Návrh doménového modelu

V doménovém modelu zaznamenávám vztahy mezi entitami, které se objevují ve výše popsaných systémových požadavcích, a dále zavádím další entity, které ač se v systémových požadavcích přímo nevyskytují, jsou potřeba k vytvoření funkčního modelu, který odpovídá reálnému použití analyzovaného softwaru.

Následující text popisuje a vysvětluje diagram doménového modelu, viz obrázek 5.1, který naleznete na konci této kapitoly.

Stěžejní částí systému je zachycení vztahu mezi zaměstnancem a zaměstnavatelem, vztah zaměstnance k projektu a záznam o činnosti zaměstnance na projektu.

Tento vztah je definován pracovní smlouvou, která může být několika druhů v závislosti na pracovní činnosti, kterou bude zaměstnanec vykonávat. Tento vztah jsem v doménovém modelu zachytil pomocí obecného předpisu označeného jako „Contract“, který nabývá specifických vlastností až vybranou konkrétní implementací předpisu. Tento způsob jsem zvolil jelikož druhy smluv jsou pevně dané a dlouhodobě zůstávají stejné.

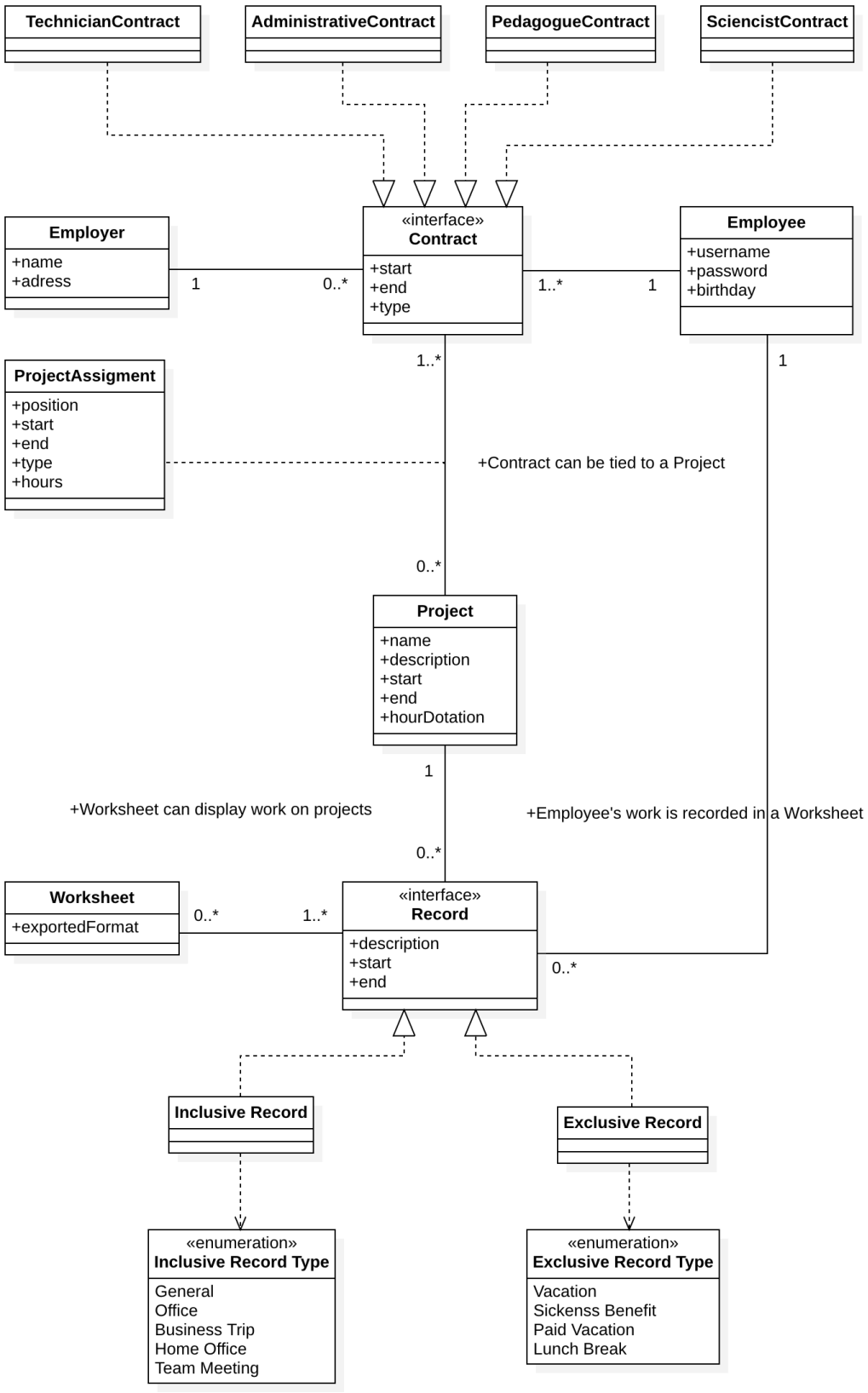
Při návrhu vztahu mezi zaměstnancem jsem zvolil variantu přiřazení projektu k vybrané pracovní smlouvě, která existuje mezi zaměstnancem a zaměstnavatelem. Důležité bylo zachytit skutečnost, že zaměstnanec se po dobu trvání smlouvy může účast několika projektů současně a jeho účast na projektu, se může dynamicky měnit v průběhu jedné smlouvy.

Nikde nenastane situace, kdy by zaměstnanec podepsal pracovní smlouvu se zaměstnavatelem a nebyl by součástí projektu. Vždy existuje minimálně jeden projekt, na kterém zaměstnanec pracuje.

Záznam o činnosti na projektu, dále jen „záznam“, je položka, která zahrnuje veškerou agendu docházky od záznamů práce na projektu, přes pracovní cesty po výběr dovolené. Podobně jako u položky „Contract“ jsem se rozhodl pro realizování pomocí obecného předpisu, který má dvě možné implementace.

Takzvaný „Inclusive Record“ jsou všechny položky, na nichž strávený čas se zahrnuje do počtu odpracovaných hodin na některém z projektů zaměstnance, kterého se týkají. Přesná specifikace tohoto záznamu je vždy určena výběrem z výčtu typů pojmenovaným „Inclusive record Type“.

Naopak „Exclusive Record“ zahrnuje všechny položky, na nichž čas strávený se zahrnuje do užívání času dovolené, jehož horní limit je stanoven v dané implementaci předpisu „Contract“.



Obrázek 5.1. Doménový model zachycující vztahy mezi entitami v docházkovém systému.

Kapitola 6
Implementace

Zvolený software, jeho architektura a komponenty vycházejí ze zadání této práce, ze získaných požadavků, výše zpracované analýzy a mých znalostí.

Jako nejvhodnější architekturu navrhuji klient server, která zajistí jednoduchou budoucí rozšiřitelnost a zároveň bude dobře použitelná při aktuálních požadavcích na využití v řádu jednotek uživatelů. Server s rozhraním REST poběží v prostředí Node.js[1]. Uživatelská data server ukládá do databáze SQLite[2]. Klient bude postaven jako single page webová aplikace s použitím javascriptového frameworku ReactJS[3].

Všechny výše zmíněné části (klient, server a databáze) budou zabaleny do samospustitelné desktopové aplikace pomocí knihovny Electron[4], která podporuje všechny hlavní platformy Windows, macOS a Linux. Aplikaci bude možné přepnout do režimu klienta, kdy se aplikace připojí na jiný server dostupný na lokální síti.

V následujícím textu popisuji, jakým způsobem probíhala implementace v předcházejícím textu návržené aplikace. Vysvětlím to, proč je kód rozdělen na několi projektů, jak jsou tyto projekty spravovány a jak jsou mezi sebou propojeny. Dále popíši, jakým způsobem jsem vytvořil algoritmus na simulování záznamů pracovníka.

V této kapitole se zaměřím na vysvětlení způsobu implementace simulačního algoritmu, který doplňuje a vytváří záznamy práce tak, jak by mohl vybraný pracovník na vybraném projektu pracovat. Popíši, kde v projektu se nachází, jaké jsou jeho vstupy a výstupy a jak funguje algoritmus samotný.

6.1 Jednotlivé projekty

Jak jsem již nastínil v úvodu kapitoly, vytvořil jsem tři samostatné projekty, které zajistí přehlednost, modularitu, budoucí udržitelnost a rozšiřitelnost kódu.

6.1.1 Aplikace

Projekt nazvaný Aplikace, k nalezení ve složce projektu */packages/application* v příloze A, obsahuje veškeré komponenty, rozvržení a logiku uživatelského rozhraní. Dále využívá API, které je zpracováno jako samostatný projekt vystavující REST rozhraní. V rámci projektu aplikace jsem vytvořil abstrakci nad jednotlivými api dotazy.

6.1.2 Api

Projekt nazvaný Api, k nalezení ve složce projektu */package/api* v příloze A, obsahuje byznys logiku celé aplikace, připojení k databázi a vystavuje RESTové rozhraní pro front endouvou část této práce. Součástí byznys logiky je především simulace zaměstnance, která zahrnuje práci s pracovními záznamy, přiřazeními do projektů a práci s kalendářem.

[1] Oficiální web Node.js. `https://nodejs.org/en/`. Aktuální k 12. 5. 2020.
[2] Oficiální web SQLite. `https://www.sqlite.org/`. Aktuální k 12. 5. 2020.
[3] Oficiální web ReactJS. `https://reactjs.org/`. Aktuální k 12. 5. 2020.
[4] Oficiální web Electron. `https://www.electronjs.org/`. Aktuální k 12. 5. 2020.

6.1.3 Databázový model

Projekt nazvaný Databázový model, k nalezení v příloze A ve složce projektu */package/domain−model*, obsahuje fasádu a implementované třídy pro práci s databázovými objekty. Dále abstrahuje připojení ke konkrétní databázi a disponuje funkcemi pro vytvoření dat pro testování a demo.

Balíček projektu jsem pojmenoval `domain-model`, přestože se jedná o implementační projekt a ne na platformě nezávislý návrh doménového modelu, jak jej definuje Martin Fowler v knize „P of EAA“. [7] Název jsem ponechal, protože zde definuji univerzální rozhraní pro ostatní projekty této práce, které je nezávislé na implementované databázi. Dále jsem zde implementoval práci s databází LiteSQL pomocí knihovny Sequelize. Díky existenci tohoto projektu, oproti implementaci databáze přímo v projektu API, lze velmi jednoduše implementovat kód pro další databáze, například samostatně bežící PostgreSQL, pokud by mnou zvolená LiteSQL nevyhovovala.

Typicky pro každý model existují tři různé fasády a jedna třída, která implementuje práci s databází. Fasády s názvem odpovídajícím doménovému modelu obsahují atributy objektů získávaných z databáze. Fasády s koncovkou *ToCreate* obsahují atributy, které jsou nutné či volitelné pro vytvoření záznamu v databázi. Fasády s koncovkou *Attributes* specifikují atributy dostupné na třídě, která umožňuje práci s databází.

Třída s názvem entity z doménového modelu s koncovkou *Model* vždy implementuje třídu s koncovkou 'Attributes'. Zároveň rozšiřuje třídu Model z knihovny Sequelize Typescript, která se stará o ORM a komunikaci s databází.

6.2 Verzování

K správě verzí projektů jsem použil Git repozitář. Jelikož jsem tuto práci rozdělil na více menších projektů, musel jsem si vybrat způsob, jakým je budu spravovat. To znamenalo ukládat všechen kód do jednoho repozitáře, způsob mono repozitář, nebo pro každý z projektů vytvořit vlastní Git repozitář, způsob poly repozitář. Případně hybridní způsob kombinace předchozích. Všechny varianty jsou vhodné pro různé druhy projektů.

Proč jsem se rozhodl používat níže zmíněné nástroje, naleznete v příloze o verzování D, kde porovnávám možné způsoby správy repozitáře nebo repozitářů a kde popisuji nástroje ke správě mono repozitáře.

Pro tuto práci jsem zvolil správu mono repozitáře pomocí nástroje Lerna, jelikož je možné začít bez nutnosti konfigurace. Dále proto, že při dvou aplikacích a jedné knihovně a při vývoji jedním programátorem není problém držet stejné nastavení příkazů napříč projekty, aby bylo možné používat globální příkazy z Lerna CLI Tool.

6.3 Algoritmus simulace

Z předcházejících kapitol plyne, že cílem práce není studium chování zaměstnanců. Proto, abych mohl simulaci provádět, zadefinuji si, jaké jsou moje představy a očekávání simulace.

- Záznamy vytvořené simulací jsou vkládány náhodně.
- Dny, ve kterých již existují záznamy a které jsou vybrány k doplnění simulací, jsou vyplněny do délky pracovní doby. Nové záznamy jsou pouze v pracovní době.
- Simulace vyčerpá dostupné neodpracované hodiny.

6.3.1 Knapsack problém, problém batohu

Po definování vstupů a výstupů jak formálně, tak neformálně se můžu podívat na to, jaké již existující problémy mají vystupy a výstupy podobné a zkusit je namapovat. Jako nejlepší kandidát podobný simulaci se jeví Knapsack problém, který v následujícím textu detailně popíši a porovnám se mnou řešeným.

Definice problému batohu podle Hanse Kellererna, „pro danou množinu $N = \{1, \ldots, n\}$ s nezápornými váhami w_j a ziskem p_j, $j = 1, \ldots, n$, a batohem o kapacitě c, problém batohu je vybrat podmnožinu množiny N takovou, jejíž součet zisků je největší a celková váha nepřesáhne kapacitu batohu c.“[8]

Na vstupu podle výše zmíněné definice je

- množina n prvků s danou váhou a ziskem,
  ze které mohu vybírat tak, abych naplnil batoh.
- kapacita c,
  která představuje maximální kapacitu batohu.

A na výstupu je

- množina prvků s danou váhou a ziskem,
  která optimálně naplňuje batoh.

Vstupy a výstupy problémů batohu a simulace, který řeším, se na první pohled mírně překrývají, jak je možné vidět v následujících tabulkách 6.1 a 6.2.

Na množinu prvků, ze kterých se u problému batohu vybírá, musí existovat způsob, jak namapovat vstupy simulace, viz tabulky 6.1 a 6.2, tak, aby každý záznam měl váhu a zisk, tedy je potřeba umět vytvořené prvky porovnávat a ohodnotit podle vybraných vlastností.

Pokud vycházím z mapování výstupů 6.2, aby simulace mohla fungovat stejně jako Knapsack problém, musím ze vstupů Simulace 6.1 vytvořit množinu nových záznamů takových, ze kterých budu moci vybrat optimální naplnění batohu. V případě simulace batohem bude velikost úvazku, tedy maximální možný počet odpracovaných hodin. Prozatím nepočítám s přesčasy.

Knapsack	Simulace
množina prvků	časové období projekt existující záznamy vlastnosti pracovníka pseudo náhodný výběr
kapacita batohu	velikost úvazku

Tabulka 6.1. Porovnání vstupů problémů Knapsack a Simulace.

Knapsack	Simulace
vybraná množina prvků	nové záznamy

Tabulka 6.2. Porovnání výstupů problémů Knapsack a Simulace.

Množinu nových záznamů pro vstup můžu vytvořit pomocí množiny všech přípustných záznamů dle vstupů simulace. Jejich váhou je doba trvání. Ovšem jejich zisk, již není na první pohled zřejmý.

Pokud nemám žádné preference mezi záznamy, ohodnotím je všechny stejně hodnotou $w_j = x$, kde $x \in \mathbb{R}$. To vyhovuje simulaci jen v případě, že implementace řešení problému batohu bude záznamy vybírat náhodně.

Pokud bych preferoval doplnění existujících záznamů, poté se již hodnocení záznamů liší. Záznamy ohodnotím následujícím způsobem hodnotou $w_j = y$, kde $y \in \{1, 2\}$. Kde zisk *2* mají záznamy, které budou do stejného dne jako již existující. Ostatní mají zisk *1*.

Jelikož jsem nenašel jiný vhodnější způsob hodnocení záznamů, než je nehodnotit, tedy přiřadit všem stejný zisk, což problém batohu redukuje na jednodušší, nepoužil jsem řešení Knapsack problému k provádění simulace.

6.3.2 Obecné vstupy a výstupy

V následujícím textu popíši dostupná data na vstupu simulace a očekávaný výstup obecně bez závislosti na zvolené implementaci.

Parametry na vstupu simulace zahrnují, kdy se má simulace odehrávat, koho simulujeme a co už se stalo. Konkrétní popis jednotlivých parametrů následuje.

- Časové období
  definuje rozmezí, od kterého dne do kterého dne můžou být vytvářeny nové záznamy. Toto období může být mimo nebo překrývající se s vybraným úvazkem na projektu. Výsledkem simulace poté budou žádné nebo takové záznamy, které splnily i dalších zadaná kritéria.
- Velikost úvazku
  definuje jakým způsobem je pracovník zaměstnán, zda na plný, poloviční nebo jiný částečný úvazek. Je vyjádřen číselnou hodnotou z oboru reálných čísel v intervalu $<0,\infty)$. Algoritmus nezohledňuje zákony, viz kapitola 2, a zpracuje i vstupy s hodnotami 1,5 nebo 2 a více úvazku.
- Projekt, přiřazení do projektu
  specifikuje období projektu, kdy pracovník mohl pracovat. To může být rozdílné od období projektu jako celku.
- Existující záznamy
  je množina pracovních záznamů, kdy zaměstnanec pracoval nebo čerpal dovolenou. Určuje pro algoritmus časové intervaly, kdy není možné vytvářet nové záznamy.
- Vlastnosti pracovníka
  algoritmem zpracovatelná informace o tom, jak se pracovník chová. Kdy začíná pracovat, kdy končí s prací, které dny v týdnu pracuje.
- Pseudo náhodný výběr
  slouží k výběru dní, ve kterých budou nové pracovní záznamy vytvořeny v případě, že dnů splňující kritéria pracovníka, viz předchozí bod vlastnosti pracovníka, je více než čas, který zbývá odpracovat do splnění úvazku na projektu. Tento výběr zajistí například pseudo náhodný generátor čísel. Případně se nemusí jednat o náhodný výběr, pokud chci upravit vlastnosti simulace.

Výstup simulace je pouze jeden, viz následující seznam.

- Nové záznamy
  vytvořené simulací odpovídající náledujícím parametrům
  - nepřekrývající se s existujícími před simulací
  - existující pouze ve zvoleném časovém období
  - existující pouze v období zaměstnanecké smlouvy
  - existující pouze v období přiřazení do projektu
  - existující pouze v období projektu
  - odpovídají vlastnostem pracovníka
    - existují pouze ve vybrané pracovní dny
    - začínají ve zvoleném časovém rozmezí nebo později
    - končí před nebo ve zvoleném časovém rozmezí
  - společně s již existujícími v součtu hodin nepřesahují velikost úvazku přiřazení na projektu

6.3.3 Pseudokód algoritmu

Jelikož by celý pseudokód v jednom bloku byl značně nepřehledný, rozdělil jsem jej do několika následujích bloků. Dále v porovnání se skutečnou implementací jsem odstranil všechny pomocné proměnné a rozdělení do menších funkcí, které by opět znepřehlednili pseudokód.

Definice záznamu, selectoru a simulační funkce jsem pro přehlednost umístil do příloh tohoto dokumentu, viz F.1.

Konstanty z následující ukázky jsou hodnoty, kterými můžeme upravovat chování simulace. Nejsou přístupné uživateli a nemám předpoklad, že by se po vydání aplikace měnily. Konstanta `MONTHLY_ASSIGNED_TIME` vyjadřuje, kolik hodin měsíčně má plný úvazek, `MINIMUM_FREE_TIME_IN_DAY`, určuje, kolik zbývajícího pracovního času musí ve vybraném dni být, aby byl tento den zařazen do simulace a byli v něm vytvořeny nové simulované záznamy.

```
MONTHLY_ASSIGNED_TIME = 160 hours
MINIMUM_FREE_TIME_IN_DAY = 30 minutes
SIMULATED_RECORD_DESCRIPTION = 'Simulated record'
SIMULATED_RECORD_TYPE = 'office'
```

Výpis 6.1. Definice konstant použitých v pesudokódu.

V následující ukázce jsem vypsal proměnné, které je potřeba připravit před samotným vytvářením záznamů.

Jelikož simulace příjmá pole existujících záznamů, pro přímý přístup k záznamům podle datumu *from*, který je opakovaně potřeba, vytvářím slovník *existingRecordsDictionary*.

Zbylá práce s časem, věřím, že nepotřebuje bližší popis.

```
existingRecordsDictionary = map existing records to dates

usedWorkTime = sum of existing records time
monthlyAssignedTime = MONTHLY_ASSIGNED_TIME * extend
remainingWorkTime = monthlyAssignedTime - usedWorkTime
```

Výpis 6.2. Definice proměnných použitých v pesudokódu.

Příprava pracovních dnů je první skutečnou částí algoritmu simulace. Jedná se o přípravu vytvoření pole všech přípustných datumů, kdy by simulovaný pracovník podle jeho specifikovaných vlastností a úvazku na projektu mohl pracovat.

```
workingDates = empty array
for each (day in between from date and to date) {
  isWorkingDay = day exists in working days in a full week
  if (isWorkingDay) {
    add day to workingDays
  }
}
```

Výpis 6.3. Ukázka přípravy pracovních dnů.

Z připravených možných pracovních dnů z předchozí ukázky 6.3 v ukázce následující 6.4 z těchto dnů vybírám dny, ve kterých budou vytvořené nové simulované záznamy.

Dny jsou vybírány pomocí výběrové funkce nazvané *selector*, která je jedním ze vstupů algoritmu simulace. Díky vyčlenění způsobu výběru tímto způsobem, můžu flexibilně a přehledně měnit jádro simulace, pokud by to bylo potřeba. Zároveň pokud by algoritmus výběru byl sám o sobě komplikovaným, jako například generace pseudo náhodných čísel, viz [9], je jeho náročnost a specifika jasně odlišena od zbytku simulace. Ve výchozím nastavení je jako *selector* použita funkce *Math.random()*[1] dostupná podle definice ECMAScript 2021[10].

```
selectedDates = empty array
while (date in workingDates) and (remainingWorkTime > 0) {
  selectedDateIndex = selector(workingDates size)
  selectedDate = workingDates(selectedDateIndex)
  selectedDateExistingRecords = get same date records
                                from existingRecordsDictionary

  remainingWorkTime -= sum of selectedDateExistingRecords time

  if (remainingWorkTime >= MINIMUM_FREE_TIME_IN_DAY) {
    add selectedDate to selectedDates
  }

  workingDates remove selectedDate
}
```

Výpis 6.4. Ukázka výběru simulovaných dnů.

[1] Přehledný popis funkce `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random`. Aktuální k 10. 5. 2020.

Ve vybraných dnech z předchozí části 6.4 nyní v následující ukázce 6.5 pokračuji vytvářením záznamů. Tělo celé funkce je k nalezení v příloze, viz F.2. Následující ukázka je tělo for cyklu, který prochází každý záznam v daném pracovním dni, který byl vybrán k doplnění simulovanými záznamy. Cyklus prochází každým dnem, kde počet záznamů je alespoň jeden.

Vybrané dny, ve kterých neexistuje jedinný záznam, popisuje ukázka pesudokódu v příloze, viz F.2.

```
if (previous record exists) {
  isRecordInBetween =
    (record from time) > (previous record to time + minute)
}

isRecordFromAfterWorkStart =
  (record from time) > (working hours begininng time)
isRecordToBeforeWorkEnd =
  (record to time) < (working hours end time)

if ((record is first) && isRecordFromAfterWorkStart) {
  newPartialRecord = new Record(
    employee, proiject id, date,
    working hours begininng time,
    (record from time) - minute,
  )
  add newPartialRecord to newRecords
}

if (isRecordInBetween) {
  newPartialRecord = new Record(...,
    (previous record to time) + minute,
    (record from time) - minute,
  )
  add newPartialRecord to newRecords
}

if (record is last) {
  newPartialRecord = new Recrod(...,
    (record to time) + minute,
    working hours end time,
  )
  add newPartialRecord to newRecords
}
```

Výpis 6.5. Ukázka tvorby simulovaných záznamů.

6.3.4 Umístění simulace v projektu

Funkce pro spuštění simulace je umístěna v projektu API, soubor k nahlédnutí v příloze A v */packages/api/Simulation/runSimulation.ts*. Krom toho hlavního bodu, ve složce *Simulation* jsou další pomocné funkce přímo související se simulací.

Simulaci z API lze spustit pro konkrétního uživatele pomocí dotazu typu POST na endpoint */employee/ : id/simulation/run*. Příklad volání v unixovém operačním systému může vypadat například takto, viz 6.6.

```
curl -request POST \
  -url http://api/employees/3/simulation/run \
  -header 'content-type: application/json' \
  -data '{
    "projectAssignmentId": "3",
    "from": "2020-01-01T00:00:00.000Z",
    "to": "2020-01-31T00:00:00.000Z"
}'
```

Výpis 6.6. Volání endpointu pro spuštění simulace pomocí curl.

V příkladu jsou zobrazeny všechny povinné atributy pro úspěšné volání simulace. Po dokončení simulace je vrácen HTTP status 200 OK. Běh simulace není časové náročný a proto je dotaz na server otevřený po celou dobu průběhu simulace.

6.3.5 Implementačně závislé vstupy a výstupy

V následující části popisuji konkrétní implementaci algortimu simulace, která se nachází ve funkci *runSimulationHelper* v projektu API, */packages/api/Simulation/runSimulation.ts*. Následuje definice funkce.

```
async function runSimulationHelper(
    records: Record[],
    projectAssignment: ProjectAssignment,
    employee: Employee,
    from: Date,
    to: Date,
): Promise<RecordToCreate[]>;
```

Výpis 6.7. Definice funkce vykonávající simulaci.

Následuje seznam implementačně závislých vstupů simulace.

- records
  existující záznamy načtené z databáze.
- projectAssignment
  přiřazení do projektu. Určuje projekt a vymezuje maximum od kdy do kdy se mohou záznamy vytvářet.
- employee
  informace o zaměstnanci, kterého simulujeme. Obsahuje detaily jeho chová, viz obecné vstupy popsané výše.
- from
  určení dne, od kdy se simulace provádí. Včetne zadaného dne.
- to
  určení dne, do kdy se simulace provádí. Včetně zadaného dne.

Následuje seznam implementačně závislých výstupů simulace.

- recordsToCreate
  nově vytvořené záznamy ve formátu pro zápis do databáze.

Kapitola 7
Testování

V této kapitole se zaměřím na testování uživatelského rozhraní a to především na testování bez uživatelů. Tento způsob volím také z důvodů probíhající světové pandemie[1]. Dále jelikož setkávání se mimo nezbytně nutné není doporučeno a jelikož uživatelské rozhraní lze testovat bez uživatelů. Ale také jelikož pro projekt ve stavu vývoje je tento způsob jednodušší, není potřeba připrava testovacího balíčku a dalších materiálů. [11]

Samozřejmě je možné se s uživateli spojit pomocí online komunikačních prostředků, bohužel to pro tento projekt není vhodné a bylo by technicky náročně. Nejedná se o webovou službu, kterou by bylo možné jednoduše otevřít webovým prohlížečem uživatele. Ale jedná se o desktopovou aplikaci, kterou by si sami uživatelé museli nainstalovat. Bohužel testovacího sestavení aplikace a distribuční kanály s tím spojené, nebyli součástí mého plánu pro implementaci.

Po testování uživatelského rozhraní se zaměřím na to, co uživatelé nevidí, ale je neméně důležité otestovat pro dobrý uživatelský zážitek. Popíšu, jakým způsobem jsem testoval kód aplikace, na které části jsem se zaměřil a jaký software jsem použil.

Dle informací uvedených v příloze E jsem se rozhodl použít pro testování kódu knihovnu Jest, která nabízí množství nástrojů pro unit a integrační testování, které používám, jelikož se jedná o standard při testování softwaru.

7.1 Heuristické hodnocení UI

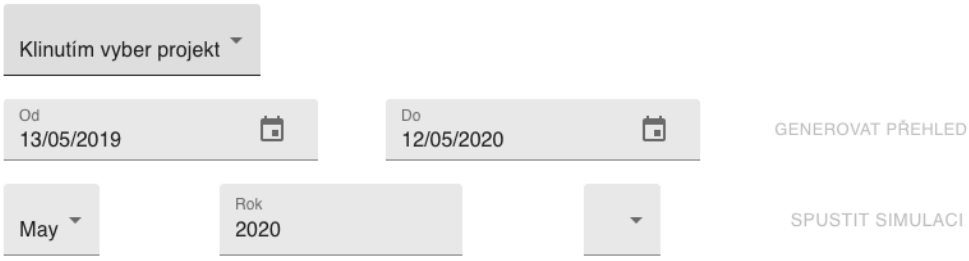
Přestože heuristické hodnocení může být silně ovlivněno osobami provádějící testování a s nižším počtem osob se množství odhalených problémů snižuje, stále se jedná o validní a chyby použitelnosti odhalující metoda. I za skutečnosti, že vyhodnocení provádím sám, předpokládaný počet odhalených problémů se v procentuálním vyjádření pohybuje mezi 20 a 51 %. [12]

Seznam heuristik používám podle Jakoba Nielsena, viz jedna z jeho novějších verzí. [13]

7.1.1 Simple and natural dialog

„Dialogy by neměli obsahovat informace, které jsou nerelevantní nebo zřídka potřebné. Každá část informace navíc v dialogu zápasí s relevantní částí informace a zeslabuje její relativní viditelnost. Všechny informace by se měli objevit v přirozeném a logickém pořadí.“ [13]

[1] Šíření onemocnění COVID-19



Obrázek 7.1. Ukázka detailu zaměstnance s zobrazením nerelevantních tlačítek.

Na obrázku 7.1 můžete vidět ukázku, kterou podle heuristik hodnotím jako chybu. Akční dialogy pro generování přehledů a pro spouštění simulace by měli být schované, pokud není vybraný žádný projekt.

7.1.2 Speak the user's language

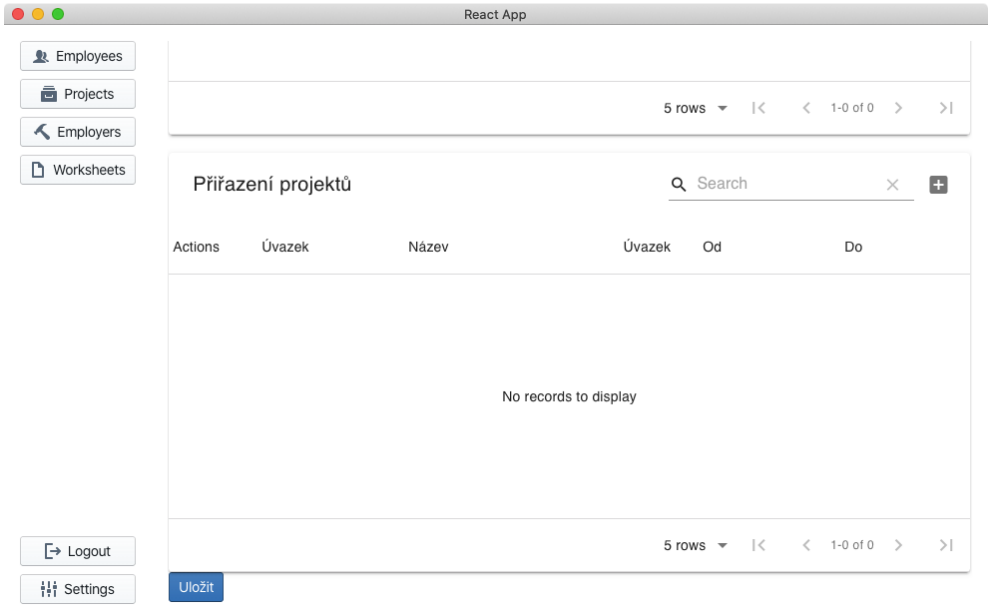
„Dialog by měl být vyjádřen jednoduše slovy, frázemi a koncepty známými pro uživatele, raději než systémově zaměřenými názvy.“ [13]

Pokud se znovu podívám na obrázek 7.1, z pohledu jazyka uživatele pozitivně hodnotím nepoužité generického *Generovat* nebo *Simulovat*, ale přesnější *Generovat přehled* a *Spustit simulaci*. Nedokáži ovšem posoudit, zda *generovat* a *simulovat* je dostatečně rozdílné pro cílové uživatele, aby nedocházelo k záměně. Možným řešením by byl popis *Uložit přehled* nebo *Generovat PDF*.

7.1.3 Minimalize the user's memory load

„Uživatel by si neměl muset pamatovat informace z jedné části dialogu v další části. Instrukce pro použití systému by měly být viditelné nebo jednoduše vyvolatelné kdykoli potřeba.“ [13]

Stránku přidání nového uživatele jsem navrhoval s touto heuristikou na paměti, viz wireframe G.15 v příloze. I v implementační části jsem se držel návrhu wireframů, přesto jsem udělal několik změn, které nyní zhodnotím.



Obrázek 7.2. Ukázka formuláře nového zaměstnance v malém okně.

Na obrázku 7.2 je vidět, že při přibližně při výšce 720 pixelů, je viděl pouze jedna z dvou tabulek obrazovky nového zaměstnance. Což zvyšuje náročnost na paměť uživatele, jelikož tabulka smluv a následující tabulka přiřazení do projektů jsou úzce propojené. Což je přibližně výška okna u druhého nejvyskytovanějšího rozlišení.[1] Při výšce okna přibližně 1080p, což odpovídá prvnímu nevyskytovanějšímu rozlišení, ze stejné statistiky jako předchozí, jsou již obě tabulky vidět a požadavek heuristiky jsem splnil.

7.1.4 Consistency

„Uživatelé by neměli muset přemýšlet, zda různá slova, situace nebo akce znamenají to samé.“ [13]

Všechny implementované akce jsou popsány rozlišně a neexistují duplicity. Jediná mnou nalezená inkonzistence je v zobrazování úvazků na projektu, kde ve formuláři pro spuštění simulace je zobrazen název úvazku na projektu, ale v tabulce úvazků na projektu tento název chybí.

7.1.5 Feedback

„Systém by měl vždy informovat uživatele o tom, co se děje, použitím odpovídající zpětné vazby v rozumném čase.“ [13]

Krom běžného disablování tlačítek při nedostupných akcí jsem v aplikaci neimplementoval loadovací animace. Což by mohlo při zmonohonásobení množství dat způsobit potíže s neinformovaností uživatele, že se něco děje. Nicméně při testování na nevýkonném kancelářském osobním počítači[2], jsem nezaznamenal žádné problémy s prodlevami v reakcích aplikace.

7.1.6 Clearly marked exits

„Uživatelé často volí funkce systému omylem a budou potřebovat jasně označený „únikový východ“ k opuštění nechtěného stavu bez nutnosti jít skrze rozsáhlý dialog.“ [13]

V jakémkoliv stavu aplikace je vždy přítomné hlavní menu na levé straně okna, které, si myslím, je jasným ovládacím prvkem, který uživatele dostane z jakékoliv nechtěné situace. Jak menu vypadá, lze vidět například na obrázku 7.2. Jediné, co by uživatelé mohli postrádat, je skutečné tlačítko zpět nebo křížek, ovšem myslím si, že vzhledem k již zmíněnému menu a velmi malému zanořování na jednotlivých stránkách aplikace, by tyto prvky byly přebytečné.

7.1.7 Shotcuts

„Urychlovače – neviditelné pro nováčky – často můžou urychlit interakci expertů systému tak, že systém obslouží jak znalé, tak neznalé uživatele.“ [13]

V této aplikaci jsem neimplementoval žádné speciální klávesové či jiné zkratky. Dostupné jsou standardní systémové zkratky pro práci s textem a přepínání oken.

7.1.8 Good error messages

„Měli by být vyjádřeny v přirozeném jazyce (žádný kód), přesně uvést problém a konstruktivně navrhnout řešení.“ [13]

[1] Statistika výskytu desktopových rozlišení v České republice. `https://gs.statcounter.com/screen-resolution-stats/desktop/czech-republic`. Aktuální k 12. 5. 2020.

[2] Konfigurace PC, procesor Intel Pentium G3220, operační paměť 4GB DDR3 a 250GB SATA 3 SSD.

Obrázek 7.3. Ukázka chybových hlášek při vytváření nového projektu.

Na obrázku 7.3 lze vidět příklad chybových hlášek splňujících heuristiku o dobrých chybových hláškách. Samozřejmě jsem programátor a jazyk uživatelů může být jiný, ale na první pohled se rozhodně nejedná o hlášky, které by jen programaticky vypsali nesplněnou podmínku, například $dateFrom < dateTo$.

V aplikaci jsem neimplementoval chybové hlášky, které by souvisely s něčekanými chybami, jako například nespuštěný server, nebo špatně odeslaný požadavek na API, jelikož to nejsou chyby, které uživatel může ovlivnit nebo vyřešit. Server je implementovaný v aplikaci, takže se nemůže stát, že by se uživatel nepřipojil například z důvodu špatného přípojení k počítačové síti, jako tomu je například u webových aplikací.

7.1.9 Prevent errors

„Ještě lepší než dobré chybové hlášky je pečlivý design, který v první řadě zabrání vzniknutí problému.“ [13]

Pokud se ještě jednou odkáži na ukázku tvorby nového projektu, viz 7.3, různé druhy vstupů, zabraňují uživatele vložit hodnoty jiného typu, čímž předcházím možným chybám, jako je vložení textu do vstupu pro časovou dotaci, nebo vložení neexistujícího data do vstupu pro datum zahájení projektu.

7.1.10 Help and documentation

„Přestože je lepší, pokud systém lze používat bez dokumetace, může být nezbytné poskytnout pomoc a dokumentaci. Každá taková informace by měla být lehce nalezitelná, měla by se zaměřovat na úkol uživatele, měla by obsahovat list konkrétních kroků k provedení a němela by být moc dlouhá.“ [13]

Aktuálně neexistuje žádný dokument s dokumentací a návody pro tuto aplikaci, pokud pominu tento dokument, který obsahuje informace o tom, jak jak by aplikace měla

fungovat podle návrhu. Zároveň tento dokument nepovažuji za uživatelskou dokumentaci, protože to není, ale také nespňuje požadavky na jednoduchost a délku.

7.1.11 Shrnutí hodnocení podle heuristik

V celém předchozím textu s hodnocením uživatelského rozhraní podle heuristik jsem slovně popisoval nalezená pozitiva i chyby, na které jsem se zaměřil více, jelikož se dají jednodušeji specifikovat v porovnání s vypisováním pozitiv.

Závažnost chyb	Slovní význam závažnosti
Neaplikovatelné	Heuristika není aplikovatelná vzhledem k druhu aplikace nebo projektu.
Vysoká	Implementace vůbec nesplňuje doporučení heuristik.
Střední	Implementace částečně splňje doporučení.
Nízká	Implementace splňuje doporučení.

Tabulka 7.1. Stupnice hodnocení podle heuristik.

Podle tabulky 7.1 převádím v následující tabulce 7.2 převádím slovní hodnocení a popis na stupnici závažnosti chyb.

Heuristika	Závažnost chyby
Simple and natural dialog	Střední
Speak the user's language	Nízká
Minimalize the user's memory load	Nízká
Consistency	Nízká
Feedback	Vysoká
Clearly marked exits	Nízká
Shotcuts	Střední
Good error messages	Střední
Prevent errors	Nízká
Help and documentation	Vysoká

Tabulka 7.2. Přehled závažnosti chyb heuristik.

Z tabulky 7.2 plyne, že před produkčním nasazením aplikace, musím opravit feedback a vytvořit dokumentaci aplikace. Další tři oblasti musím pečlivě zkontrolovat. A zbylých pět je již připravených. Samozřejmě, pokud přidám, či změním funkcionalitu uživatelského rozhraní je potřeba provést vyhodnocení pomocí heuristik znova. Také nesmím zapomenout, že vyhodnocení je doporučeno provádět ve více než jedné osobě. [12]

7.2 Testování s uživateli

Přesto, že jsem musel testování s uživateli omezit, viz zažátek této kapitoly 7, připravil jsem si krátký testovací scénář s několika body a vyzkoušel jej s několika uživateli. Ačkoli se nejednalo přesně o budoucí uživatele, myslím si, že se stále jedná o užitečné testování.

Scénář jsem postavil tak, aby zahrnoval základní a pravděpodobně nejpoužívanější funkce aplikace. Takto vypadá uživateli prováděný scénář.

7.2.1 Shrnutí testu

Uživatel se přihlásí, vytvoří nového zaměstnance, spustí na něm simulaci a vygeneruje přehled.

7.2.2 Popis testu

1. Uživatel se příhlásí do aplikace.
2. Uživatel vytvoří nového zaměstnance, dle testovacích dat, viz 7.2.3.
3. Uživatel spustí simulace na zaměstnanci vyvořeném v přechozím kroku. Parametry simulace jsou v testovacích datech, viz 7.2.3.
4. Uživatel vygeneruje přehled za období dle testovacích dat, viz 7.2.3.

7.2.3 Testovací data

Uživatelské jméno	test1
Uživatelské heslo	Heslo1
Jméno zaměstnance	Tom Been
Titul zaměstnance	Ing.
Datum narození zaměstnance	25. 3. 1985
Začátek pracovní doby zaměstnance	8:00 – 8:20
Délka pracovní doby zaměstnance	7:50 – 8:15
Pracovní dny v týdnu zaměstnance	Pondělí, Úterý, Pátek a Sobota
Typ úvazku zaměstnance	Vědec
Velikost úvazku zaměstnance	Plný úvazek
Délka úvazku zaměstnance	1. 1. 2020 – 31. 12. 2020
Název projektu zaměstnance	Projekt 1
Úvazek zaměstnance na projektu	Plný úvazek
Délka úvazku zaměstnance na projektu	1. 6. 2020 – 31. 12. 2020
Simulované období	1. 7. 2020 – 31. 7. 2020
Období generovaného přehledu	Celý rok 2020

7.2.4 Očekávaný výsledek

Na testovaném počítači je uložený soubor s přehledem vytvořeného zaměstnance se simulovanými záznamy.

7.2.5 Průběh a poznatky z testování

Uživatelé přistupovali k testovacímu počítači se spuštěnou aplikací. Přihlášení do aplikace nebyl v žádném případě problém. U vytváření zaměstnance se projevila chyba odhalená již pomocí heuristického testování, viz 7.1.3. Žádného z testerů nenapadlo zvětšit výchozí velikost okna aplikace, i když tato funkce dostupná je a z diskuze po testu uživatelé o této systémové funkci vědí.

Dalším problémem, který jsem také nastínil v heuristickém testování, viz 7.1.2, bylo překvapení z dotazu na ukládání generovaného přehledu. Tlačítko *generovat* dostatečně nevysvětluje, že se bude něco ukládat do počítače uživatele.

V ostatních bodech testu žádný problém nebyl. Testeři měli minimálně středoškolské vzdělání a několikaletou praxi s kancelářkou prací na počítači.

7.3 Unit testy

Jak již jsem zmínil výše, testování jsem provedl jen na části aplikace, konkrétně v projektu API u simulačních funkcí. Kód testů jsem uložil ve složkové struktuře do */packages/api/test* a následně do podsložek *unit* a *integration*, k nalezení v příloze A.

Pomocí unit tesů jsem zkontroloval funkce ze souboru *selectDates.ts* a *runSimulation.ts*.

7.3.1 Testy výběru datumů

Všechny testy úspěšně procházejí. Následuje list testovaných případů výběru datumů.

- funkce createWorkingDays
  - should select mondays in given period
  - should not select any days because from and to is the same date
  - should not select any days because there are non in given period
  - should not select any days because no working days are given
- funkce selectDateBySelector
  - should select dates with given non-zero amount of free time
  - should select dates with given non-zero amount of free time (no records)
  - should not select any dates with given zero free time
  - should not select any dates under free day time limit
- funkce isWorkingDay
  - should return false for empty working days array
  - should return true for Monday dates and false for everything else
  - should return true for Tuesday to Friday dates and false for weekend days
  - should return false for array including values only outside of range <0,6>
  - should return true for Wednesday and Friday dates ignoring array values outside of range <0,6>

7.3.2 Testy tvorby záznamů

Všechny testy úspěšně procházejí. Následuje list testovaných případů tvorby simulovaných záznamů.

- should create record from existing to the end
- should create record from the beginning to the existing
- should create record in between records
- should create records around one existing
- should create records around the existing records
- should create no record existing fills whole day

7.4 Integrační testy

Zde jsem se zaměřil především na komplikované databázové dotazy a filtrování databázových výsledků, které slouží jako vstupy pro algoritmus simulace.

Při spuštění testů testovací nástroj vytvoří SQLLite databázi přímo v operační paměti. Tato databáze je před každým jednotlivým testem naplněna testovacími daty a po každém jednotlivém testu jsou kompletně odstraněny všechny záznamy a databázová schémata.

7.4.1 Příprava testovacích dat

Testovací data jsou připravena jako plain javascriptové objekty. Všechny interfacy databázového modelu s konovkou *Attributes* jsem připravil rozšiřující interfacy s koncovkou *DemoAttributes*. Výhodou tohoto řešení jsou extra atributy s koncovkou $DEMO\_ID$, které specifikují provázání před vytvořením databázových cizích klíčů. Lze tedy jednoduše připravit komplikovaná provázaná data bez identifikátorů vytvářených databází. Následuje ukázka rozšířeného interfacu.

```
interface ContractDemoAttributes extends ContractAttributes {
  demoId: CONTRACT_DEMO_ID;
  employeeDemoId: EMPLOYEE_DEMO_ID;
}
```

Výpis 7.1. Ukázka rozšíření modelu pro tvorbu testovacích dat.

Připravená testovací data obsahují demo zaměstnance, simulační data, smlouvy, projekty, přiřazení do projektů a několik pracovních záznamů.

7.4.2 Seznam testů

Následuje seznam testů funkce $findContractByProjectAssignmentId$.

- should get contract by given assignment id
- should not get any contract by non existing id

Následuje seznam testů funkce $getRecordsByProjectIdAndEmployeeId$.

- should get records in given period ordered ascending by from
- should not get any records

Všechny integrační testy úspěšně fungují.

Kapitola 8
Závěr

Práci jsem zahájil seznámením se s problematikou vytváření dokumentů docházky, následovala analýza agendy, ze které vyplynula analýza požadavků s přihlédnutím na pracovníky ČVUT FEL. Dále jsem pokračoval rozborem a porovnáním existujících řešení. Po vyhodnocení skutečné potřeby nového systému jsem navrh doménový model problému a zahájil jsem práci na případech užití, které jsou doplněny wireframy uživatelského rozhraní. Následovala příprava prostředí pro implementaci a zvolení vhodných nástrojů a technologií. Poté jsem analyzoval existující algoritmické problémy podobné simulování docházky. Z této analýzy plyne použití poznatků z problému batohu. Na závěr jsem otestoval uživatelské rozhraní pomocí heuristického hodnocení a omezeného uživatelského testování vzhledem k situaci ve světě[1]. Testování algoritmu jsem provedl pomocí standardních unit a integračních testů procházejících bez chyb. Tím jsem splnil všechny požadavky zadání bakalářské práce.

8.1 Analýza agendy docházky

Analýzu agendy docházky jsem začal pochopením využití dat docházky ze strany zaměstnavatele. Jelikož tento jednoduchý problém pro zaměstnance, který si zaznamenává čas od, do a popis práce, se z pohledu zaměstnavatele stává mnohem komplexnější a údajů, které je třeba uchovávat je mnohonásobně více. A ještě více je jich pro systém, který má tyto záznamy simulovat, to je možné vidět v kapitole o implementaci algoritmu 6.

8.2 Analýza požadavků

Osobně jsem agendu docházky znal jen ze strany zaměstnance, tedy bylo důležite správně pochopit nejen uchovávaná data, ale také k čemu se data používají, tedy exportování přehledů, které například musejí, podle požadavků, obsahovat anonymizované odpracované hodiny projektů, které se netýkají daného přehledu.

Nejzákladnější požadavky na systém jsou práce se zaměstnanci, projekty, záznamy práce a simulace, která, ačkoliv je v jednom požadavku, byla nejvíce složitá na implementaci.

Na základě získaných požadavků jsem našel dvě existující řešení. Bohužel ani jedno nesplňovalo všechny požadavky a úprava pro splnění požadavků byla neexistující nebo vyžadovala rozsáhlý zásah do existujícího systému.

8.3 Implementace

Na základě prozkoumaných existujích řešení jsem se rozhodl, že je potřeba vytvořit nový systém. Zvolil jsem desktopovou aplikace, která obsahuje server s RESTovým API,

[1] Šíření onemocnění COVID-19

nenáročnou SQL databází a klienta s uživatelským rozhraním. Stejným způsobem, jako je uvnitř rozdělena aplikace, jsem vytvořil tři samostatné projekty. Aplikaci s UI, server s API a knihovnu pro práci s databází. Toto rozdělení dělá aplikaci velmi udržitelnou a flexibilní pro budoucí rozvoj. Velmi jednoduše lze aplikaci začít využívat pouze jako klienta a serverovou část skutečně spustit například v cloudu. Stejně jednoduše lze mnou vybranou databázi nahradit robustním řešením s odlišnou implementací.

S velkou překážkou jsem se setkal při implementování uživatelského rozhraní, když jsem zjistil, že knihovna komponent, kterou jsem si vybral pro uživatelsky přívětivou práci s tabulkovými daty, pracuje v kódu s daty po sloupcích. Což se i po několika pokusech ukázalo jako absolutně nepoužitelné pro pracovní záznamy.

Algoritmus simulace byl náročný především ze strany studia existujících řešení. Samotná implementace podle předpřipraveného pseudokódu již probíhala jednoduše. Velmi zjednodušeně algoritmus si podle zadaných parametrů zaměstnance a projektu vybere dny, kdy by zaměstnanec mohl pracovat. Následně danou funkcí výběru, aktuálně implementovaný je pseudo náhodný výběr, vybere dny, kdy se budou vytvářet simulované záznamy. Poté projde vybrané dny a doplní záznamy mezi existující, podle parametrů simulace, zaměstnance a projektu.

8.4 Testování

Testování jsem provedl ve dvou fázích. První bylo testování uživatelského rozhraní, kdy jsem byl situací omezen v testování s uživateli, tedy jsem navíc využil metod testování bez uživatelů, konkrétně heuristického hodnocení. Podle obou způsobů jsem našel několik málo chyb. Velmi mě potěšilo, že některé zjistění heuristického hodnocení, které tím, že jsem jej vyhodnocoval sám, bylo také omezeno, se objevily také při testování s uživateli.

Druhou fází testování byly jednotkové a integrační testy, kterými jsem prověřil funkčnost algoritmu simulace, jakožto nejkritičtější části aplikace. Celkem přes dvacet různých testů pokrývajících krajní hodnoty i běžné vstupy procházejí bez problémů.

Literatura

[1] Ministerstvo vnitra České republiky. *Zákon č. 262/2006 Sb.* 2006.

[2] Mashhood Sajid, Rubab Hussain a Muhammad Usman. *A conceptual model for automated attendance marking system using facial recognition*. IEEE, 2014. ISBN 978-1-4799-5421-6.

[3] ELEKON s. r. o. *Katalogový list Cloudový docházkový systém iTA*. 2019. https://mobatime.cz/wp-content/uploads/2018/01/iTA-7-190617-CZ.pdf.

[4] EFG CZ s. r. o. *Katalogový list Přístupový a docházkový systém Aktion eSeries*. 2018. https://www.aktion.cz/aktion_cs/download/katalogove-listy/eseries.pdf.

[5] EFG CZ s. r. o. *Dokumentace Aktion Práce s agendou Docházka Uživatelský manuál*. 2019. https://www.ecare.cz/download/get/656b74c1342db1d3c615026c93763424/aktion-cloud-prace-s-agendou-dochazka_v4.

[6] Object Management Group Inc. (OMG). *UML Superstructure Specification Version 2.0*. 2005. https://www.omg.org/spec/UML/2.0/Superstructure/PDF.

[7] Fowler Martin. *Patterns of Enterprise Application Architecture*. Sedumnácté vydání. Addison-Wesley Educational Publishers Inc, 2002. ISBN 0-321-12742-0.

[8] Hans Kellerer. *Knapsack, In: Kao MY. (eds) Encyclopedia of Algorithms*. Springer, New York, NY, 2016. ISBN 978-1-4939-2864-4. https://link.springer.com/referenceworkentry/10.1007/978-1-4939-2864-4_192.

[9] Dr Mads Haahr. *Introduction to Randomness and Random Numbers*. https://www.random.org/randomness/.

[10] ECMAScript community. *ECMAScript 2021 Language Specification, Draft ECMA-262*. 2020. https://tc39.es/ecma262/#sec-math.random.

[11] Morten Hertzum. *A Usability Test Is Not an Interview*. 2016. https://dl.acm.org/doi/10.1145/2875462.

[12] Jakob Nielsen a Rolf Molich. *Heuristic evaluation of user interfaces*. 1990. https://dl.acm.org/doi/10.1145/97243.97281.

[13] Jakob Nielsen. *Enhancing the explanatory power of usability heuristics*. 1994. https://dl.acm.org/doi/10.1145/191666.191729.

[14] Nicolas Brousse. *The Issue of Monorepo and Polyrepo In Large Enterprises*. 2019. https://doi.org/10.1145/3328433.3328435.

[15] Nicolas Brousse. *Why Google stores billions of lines of code in a single repository*. 2016. https://dl.acm.org/doi/10.1145/2854146.

[16] Lerna. *Documentation, Project README*. 2020.
https://github.com/lerna/lerna.

[17] Babel. *Documentation, Project README*. 2020.
https://github.com/babel/babel.

[18] npm. *npm Documentation*. 2020.
https://docs.npmjs.com/.

[19] Nx. *Nx Documentation*. 2020.
https://nx.dev/.

[20] Raphaël Benitte, Sacha Greif a Michael Rambeau. *State Of JavaScript*. 2019.
https://2019.stateofjs.com/.

[21] Facebook a Jest Community. *Jest Documentation*. 2020.
https://jestjs.io/.

[22] Mocha Community. *Mocha Documentation*. 2020.
https://mochajs.org/.

[23] Peldi Guilizzoni. *What Are Wireframes?* 2020.
https://balsamiq.com/learn/articles/what-are-wireframes/.

Příloha A
Soubory projektu

Všechny soubory projektu naleznete na přiloženém DVD nebo na repozitáři projektu.[1] DVD obsahuje kompletní exportovaný balíček repozitáře, kde byl projekt od začátku vyvíjen. Dále jsem na DVD umístil poslední verzi projektu, tedy verzi odpovídající poslednímu commitu před odevzdáním práce, tag 0.0.0. Poslední částí DVD je PDF dokument s touto prací.

A.1 Struktura DVD

Následující seznam popisuje strukturu kořenové složky DVD.

- attendance-system_export.tar.gz
  Exportovaný balíček repozitáře.
- attendance-system-master.zip
  Poslední verze projektu, tag 0.0.0.
- tex-document-master.zip
  Zdrojový kód tohoto dokumentu a wireframy.
- woldrkry_system_for_attendance_recording_generation_and_simulation.pdf
  Tento dokument ve formátu PDF.

[1] Url repozitáře `https://gitlab.fel.cvut.cz/woldrkry/attendance-system`. Aktuální k 16. 5. 2020.

Příloha B
Zkratky a pojmy

Tato příloha vysvětluje zkratky a pojmy použité v této práci. Přípony a formáty souborů zde nejsou uvedeny.

B.1 Zkratky

UML	Z anglického Unified Modeling Language je unifikovaný modelovací jazyk. V této práci používám verzi 2, viz specifikace [6].
FR	Z anglického functional business requirements. Česky funkční byznysové požadavky.
NFR	Z anglického non-functional system requirements. Česky kvalitativní byznys požadavky.
REST	Z anglického REpresentational State Transfer.
API	Z anglického Application Programming Interface.
HTTP	Z anglického Hypertext Transfer Protocol.
CLI	Z anglického Command-line interface.
GDPR	Z anglického General Data Protection Regulation.
ORM	Z anglického General Data Protection Regulation.

B.2 Pojmy

TypeScript	Typovaný programovací jazyk vyvíjený a spravovaný společností Microsoft.
Node.js	Prostředí pro spuštění JavaScriptu mimo webový prohlížeč.
SQLite	Plnohodnotná implementace relační SQL databáze.
Electron	Open source knihovna pro vytváření desktopových aplikací vyvíjená společností GitHub.
ReactJS	Open source knihovna pro vytváření uživatelských rozhraní vyvíjená společností Facebook.
syntaxe	Definovaná pravidla výrazů, typicky programovacího jazyka.
Git	Populární verzovací systém pro vývoj softwaru.
Sequelize	Knihovna poskytující ORM pro Node.js projekty.
PostgreSQL	Open source relační databáze.
POST	Metoda odeslání dat pomocí HTTP.

Příloha C
Seznam výpisů

Příloha D
Verzování

V této příloze podrobně popisuji různé druhy správy verzovacího repozitáře, různé přístup k dělení projektu na balíčky, knihovny a menší projekty. Dále popisuji, jaké jsou možnosti správy mono repozitářů, jelikož tato forma se jeví jako vhodná pro tuto práci.

D.0.1 Poly repozitáře

Poly repozitáře, také polyrepo, znamená, že existuje „*jeden zdrojový rezitář pro každou komponenetu a knihovnu*“ [14], která je součástí projektu, v mém případě této práce.

Z významných technologických firem tento způsob verzování projektů adoptoval Amazon, Netflix a Lyft. Pro příklady aktivních repozitářů zmíněných firem stačí navštívit jich stránky s open-source projekty na GitHubu. [14] [1] [2] [3]

Tento způsob zprávy je zláště výhodný v případech, kdy jednotlivé projekty na sobě nejsou úzce závislé.

D.0.2 Mono repozitář

Mono repozitář, také monorepo, je „*jeden zdrojový repozitář pro celou společnost nebo velmi rozsáhlý produkt*“ [14]. V mém případě tato práce je zmíněný rozsáhlý produkt.

Z významných technologických firem tento způsob verzování projektů adoptoval Google, Facebook, Microsoft, Twitter a další viz [14]. K náhlednutí, jak takové repozitéře vypadají, stačí navštívit stránky open-source projektů těchto společností. [4] [5]

Toto neplatí pro Google, jehož core codebase není opensource a je umístě na míru postaveném neveřejném zdrojovém repozitáři. [15]

Tento způsob je zvláště výhodný, pro projekty, které jsou na sobě úzce závislé. Což je případ této práce, kde všechny výše zmíněné projekty vždy běží společně a jsou součástí jedné desktopové aplikace.

D.0.3 Hybridní repozitář nebo repozitáře

Jedná se o případy, kdy pracujete s monorepem a následně se projekty distribuují do samostatných readonly repozitářů a naopak. Tento druh správy jsem nezvažoval pro jeho složitost samotného nastavení takového repozitáře. [14]

D.1 Správa mono repozitáře

Tuto práci implementuji a spravuji v monolitickém repozitáři, o kterém jsem psal v předchozím odstavci. Jelikož pro potřeby vývoje nepotřebuji a není praktické publikovat jednotlivé projekty. Potřeboval jsem způsob pro správu repozitáře, který mi umožní propojit jednotlivé projekty tak, jako by byly publikované.

[1] Amazon Web Services `https://github.com/aws`. Aktuální k 4. 5. 2020.
[2] Netflix Open Source Platform `https://github.com/Netflix`. Aktuální k 4. 5. 2020.
[3] Lyft `https://github.com/lyft`. Aktuální k 4. 5. 2020.
[4] React `https://github.com/facebook/react`. Aktuální k 4. 5. 2020.
[5] Babel `https://github.com/babel/babel`. Aktuální k 4. 5. 2020.

D.1.1 Lerna

Lerna je nástroj pro správu JavaScriptových projektů, v kontextu této implementace repozitářů, s více balíčky, v tomto případě projekty. [16] Tento nástroj je používán například vývojáři knihoven Babel, kde také vznikl [17], nebo vývojáři testovacího balíčku Jest.

Pro nahlednutí na praktické použití stačí nahlédnout na repozitáře výše zmíněných knihoven. [1] [2]

Lerna CLI Tool disponuje příkazy blížícími se balíčkovacím manažerům NPM a Yarn, které na pozadí spravuje napříč projekty. Dále umožňuje přidávání nepublikovaných balíčků, za podmínek, že je daný balíček součástí daného projektu a je Lernou spravován. Lerna při instalaci lokálních balíčků kontroluje verzi a k použití lokálního balíčku dojde pouze při shodě lokální a požadované verze instalačním příkazem. [16]

Za použití nástrojů NPM nebo Yarn lze docílit podobného chování, ovšem nelze instaloval nepublikovaný balíček. Lze pouze nainstaloval lokální verzi již publikovaného balíčku.[18]

Balíčky v repozitáři spravovaném Lernou jsou všechny na stejné úrovni a ke všem se přistupuje stejně. Podle dokumentace poté lze spouštět příkazy napříč balíčky, ovšem je potřeba příkazy definovat shodně ve všech dílčích projektech. [16]

D.1.2 Nx

Nx je nástroj na správu monolitického repozitáře stejně jako Lerna, ale přistupuje jiným způsobem k balíčkům, které se nacházejí pod jeho správou. Projekty dělí na aplikace ve složce *apps* a na knihovny ve složce *libs*. Cílem je přehledně oddělit samostatně spustitelný kód, myšleno spustitelné aplikace, servery, deamony a podobně, a sdílený kód, který je využíván napříč aplikacemi. [19]

Použití Nx CLI nástroje je taktéž velmi podobné zmíněnému nástroji Lerna. Ovšem již málo příkazů je podobných s balíčkovacími manažery NPM a Yarn. Nx je oproti systému Lerny postaven na kofiguraci, kde se specifikuje, jaké příkazy se mají kdy spouštět, kde systém Nx nalezne konfiguraci jednotlivých projektů a další. [19]

[1] Babel `https://github.com/babel/babel`. Aktuální k 4. 5. 2020.
[2] Jest `https://github.com/facebook/jest`. Aktuální k 4. 5. 2020.

Příloha E
Porovnání testovacích nástrojů

Podle dotazníku *State Of JavaScript 2019* jsem se rozhodoval mezi testovacím nástrojem Jest a Mocha, jelikož z tohoto dotazníků vycházely jako první a pátý nástroj, seřazení podle spokojenosti uživatelů. [20] Nástroje Cypress, Storybook a Puppeteer nemají nástroje pro unit testování, proto jsem je při výběru vynechal.

E.1 Jest

Jest je vyvíjen podle dostupných informací GitHub repozitáře od konce roku 2013[1]. Jeho podpora je zaštítěna společností Facebook, ale stále se jedná o opensource projekt, do kterého může přispět kdokoli.

Jest není pouze testovacím nástrojem pro Node.js[2], pro který jsem hledal testovací nástroj. Ale také umožňuje testovat správné renderování uživatelského rozhraní [21], což by v tomto projektu mohlo být užitečné při dalším vývoji.

E.2 Mocha

Mocha je vyvíjena podle dostupných informací GitHub repozitáře od začátku roku 2011[3] Není zastoupena žádnou korporací a je součástí *OpenJS Foundation*[4].

Mocha je testovací nástroj pro Node a webový prohlížeč. Podporuje pouze standardní unit testování. Extra funkce jako například mockování je potřeba zajistit externí knihovnou. Výchozí nastavení náhledů na průběh testů je méně přehledné v porovnání s novějším Jestem, viz E.1. [22]

E.3 Shrnutí

Oba testovácí nástroje poskytují funkcioalitu pro testování unit testů. Oba poskytují statistiky o pokrytí kódu testy. Oba používají stejnou syntaxi, klíčová slova *describe* a *it*. Jest nabízí pro mě osobně přehlednější přehled proběhlých testů ve výchozím nastavení a také jsou součástí knihovny mockovací nástroje. Pro Mocha je potřeba doinstalovat knihovna navíc. Jest umožňuje také testování UI komponent, to je vhodné pro budoucí rozvoj projektu.

[1] Detail repozitáře Jest z GitHub API. `https://api.github.com/repos/facebook/jest`. Aktuální k 12. 5. 2020.
[2] Oficiální web Node.js. `https://nodejs.org/en/`. Aktuální k 12. 5. 2020.
[3] Detail repozitáře Mocha z GitHub API. `https://api.github.com/repos/mochajs/mocha`. Aktuální k 12. 5. 2020.
[4] Více o OpenJS Foundation. `https://openjsf.org/`. Aktuální k 12. 5. 2020.

Příloha F
Pseudokód

Do této přílohy jsem uložil části pseudokódu, jejichž přítomnost v textu kapitoly 6.3, kde vysvětluji fungování simulačního algoritmu, není nezbytně nutná, ale jsou důležité pro ucelenost ukázek pseudokódů, které se ve zmíněné kapitole nachází. Pomáhají ještě detailnějšímu pochopení algoritmu a předcházejí možným nepochopením.

F.1 Definice

Následující ukázka je jednoduchý výpis definic, které jsou následně použity napříč zbylými ukázkami pseudokódu. Pro zjednodušení si je můžete představit jako definice interfaců z některého z vyšších programovacích jazyků.

```
record has attributes:
  employee identificator,
  project identificator,
  optional description,
  type,
  from datetime,
  to datetime,

selector is:
  input: maximal inclusive integer
  output: interger from 0 to input

simulation alghorithm is:
  input:  existing records,
          project assignment
            extend,
            project identificator,
          employee's
            identificator,
            working days in a full week,
            working hours length period,
            working hours begininng period,
          from date with day, month and year,
          to date with day, month and year,
  output: newly created records
```

Výpis F.1. Seznam definic referencovaných v ukázkách pseudokódů.

F.2 Funkce simulace

Ukázka těla for cyklu chybějícího v ukázce pod tímto textem, viz F.2, se nachází v kapitole 6.3.3, která popisuje celé fungování algoritmu na ukázce pseudokódu. For cyklus pod tímto textem, viz F.2 iteruje přes všechny dny vybrané k simulaci. Pokud se v aktuálním dnu nenachází žádný existující záznam, vytvoří záznam přes celý den. Pokud v aktuálním dnu záznamy existují, tato část je rozebrána v kapitole 6.3.3.

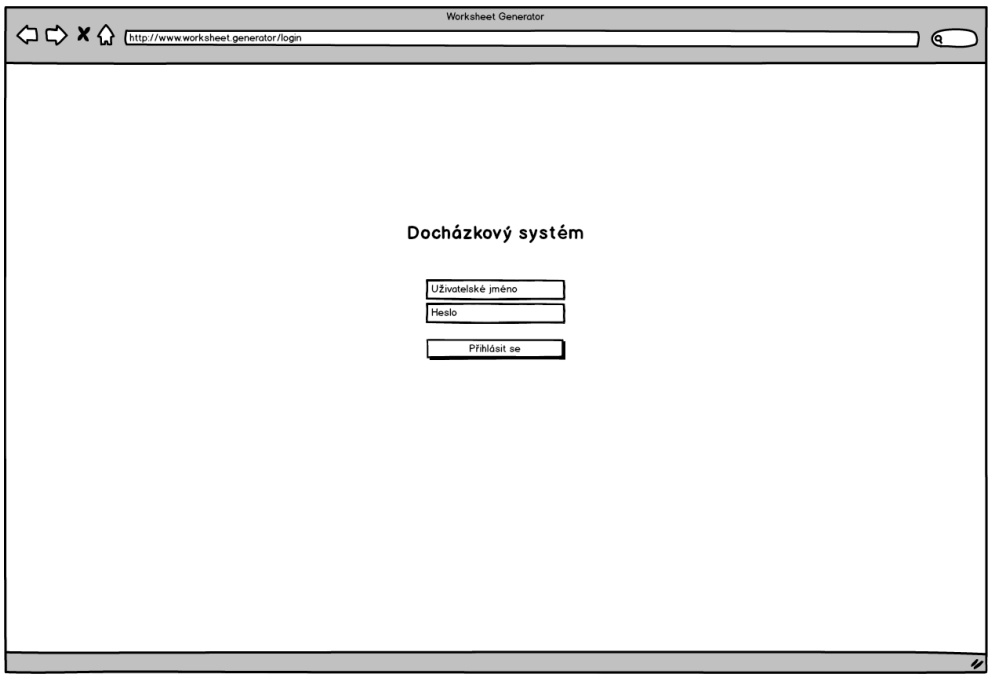
```
newRecords = empty array
for each (date in selectedDates) {
  dateExistingRecords = get same date records
                            from existingRecordsDictionary

  if (dateExistingRecords size > 0) {
    partialNewRecords = empty array
    sortedDateExistingRecords =
      sort dateExistingRecords by from date
    workingHoursEndTime =
      working hours lenght time + working hours begininng time

    for each record in sortedDateExistingRecords {
      //Pro tělo for cyklu viz text před touto ukázkou.
    }
  } else {
    fullDayRecord = new Record(
      ...,
      date with working hours begininng period time,
      date with working hours length period time,
    )
    add fullDayRecord to newRecords
  }
}

return newRecords;
```

Výpis F.2. Ukázka funkce simulace bez dnů s existujícími záznamy.

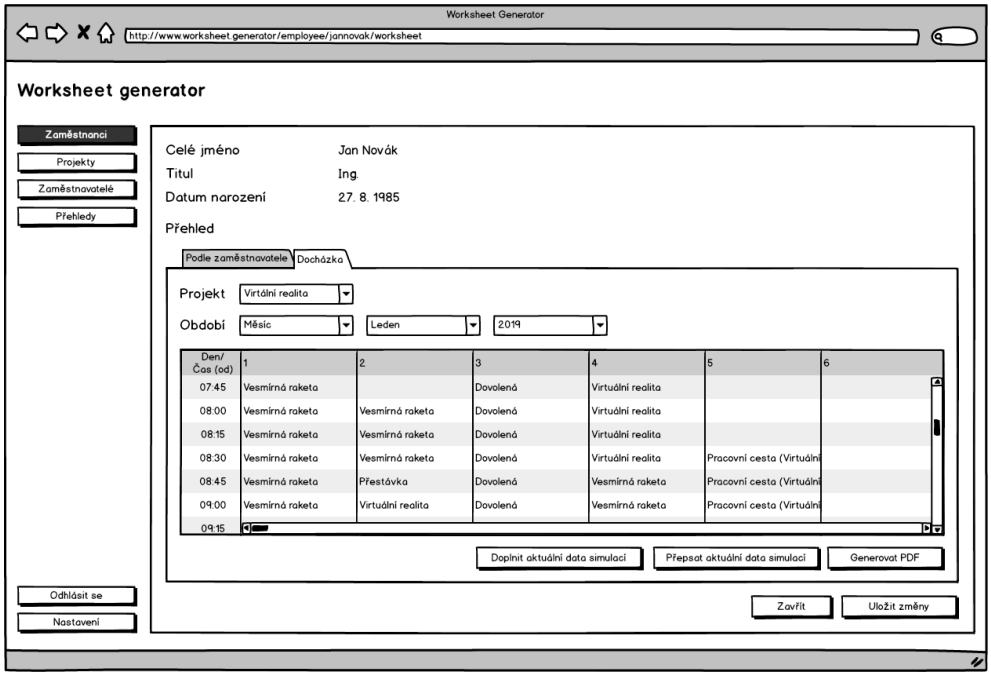
Příloha G
Wireframy

„Wireframe je schéma nebo technický plán, který pomůže vám, vašim programátorům a designerům myslet a komunikovat o struktuře vámi vytvářeného softwaru nebo webové stránky.“ [23]

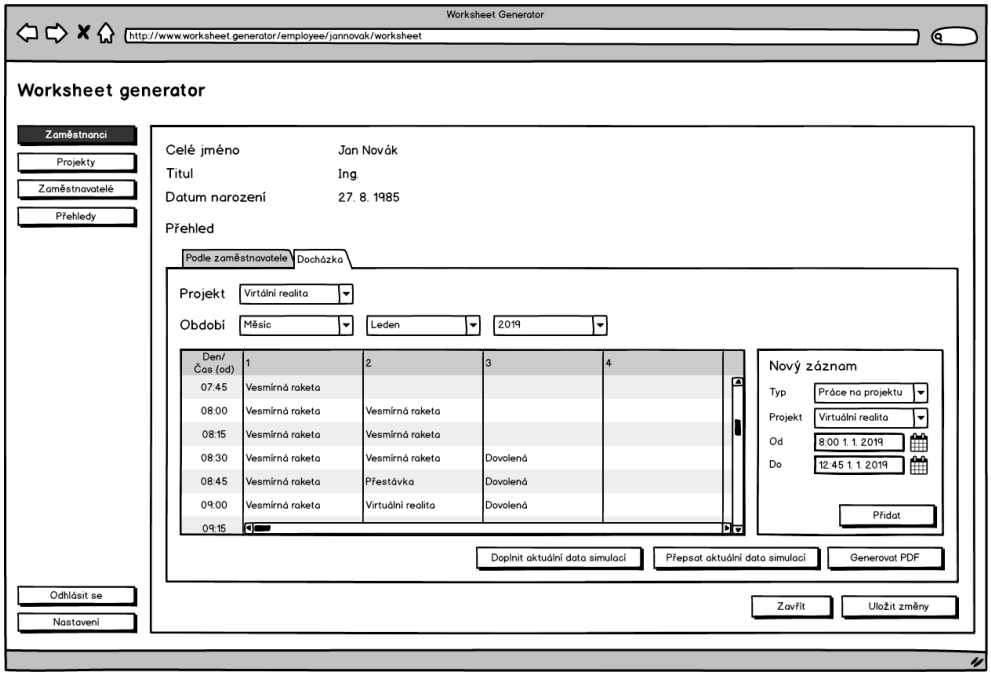
Následující wireframy zachycují podstatné akce, které budou uživatelé v aplikaci vykonávat. Důležité jsou zobrazené informace, tlačítka a prvky viditelné současně na jedné obrazovce. Nejedná se o finální návrh designu.



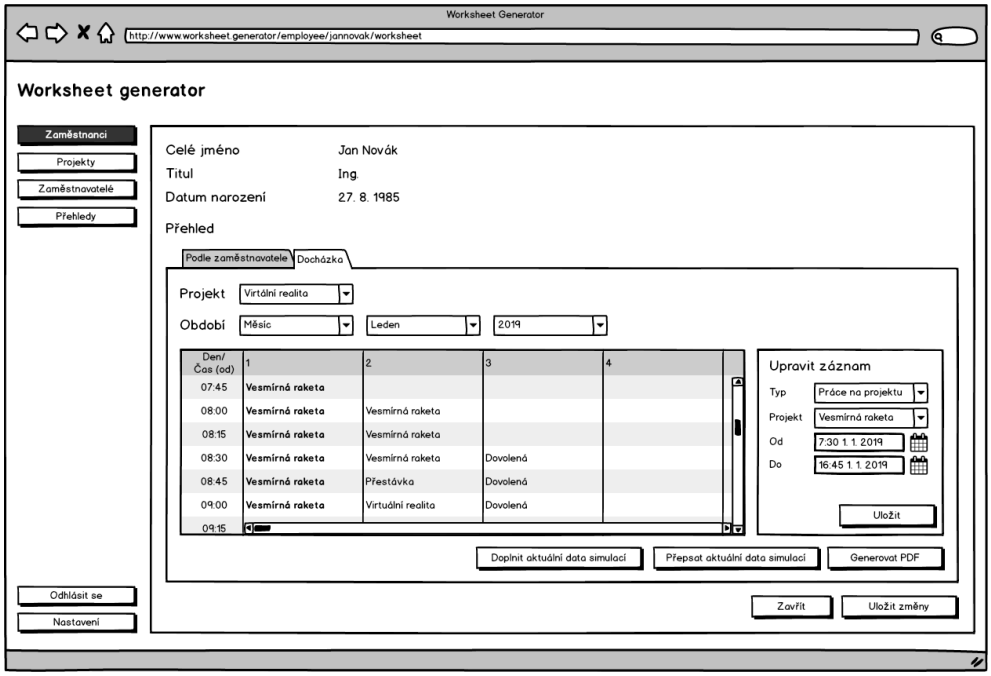
Obrázek G.1. Přihlašovací obrazovka systému.



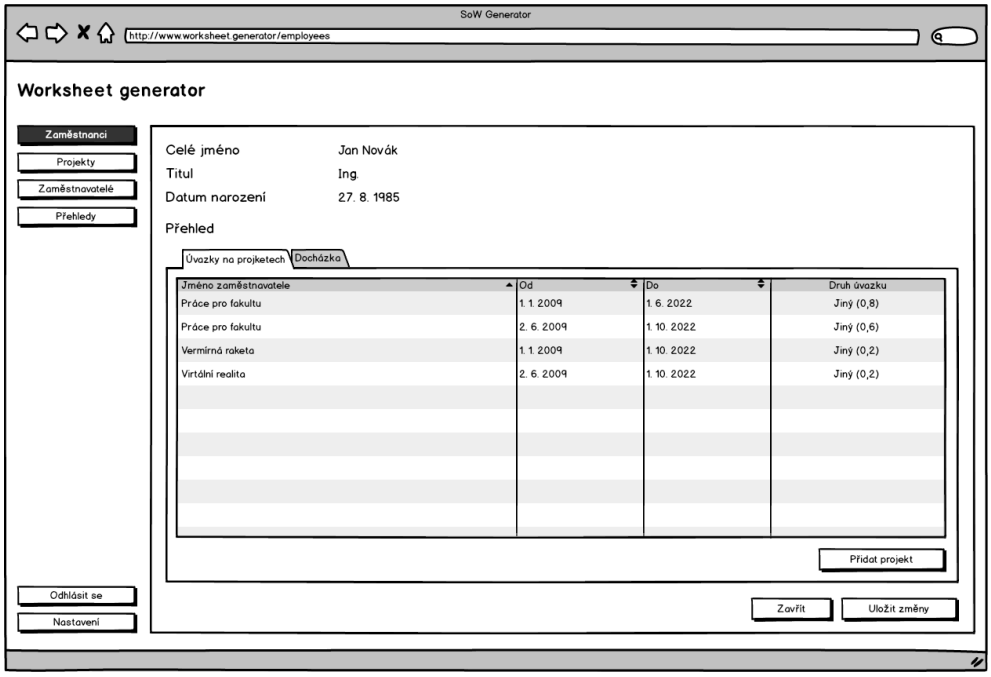
Obrázek G.2. Detail zaměstnance na záložce přehledu docházky s měsíčním zobrazením.



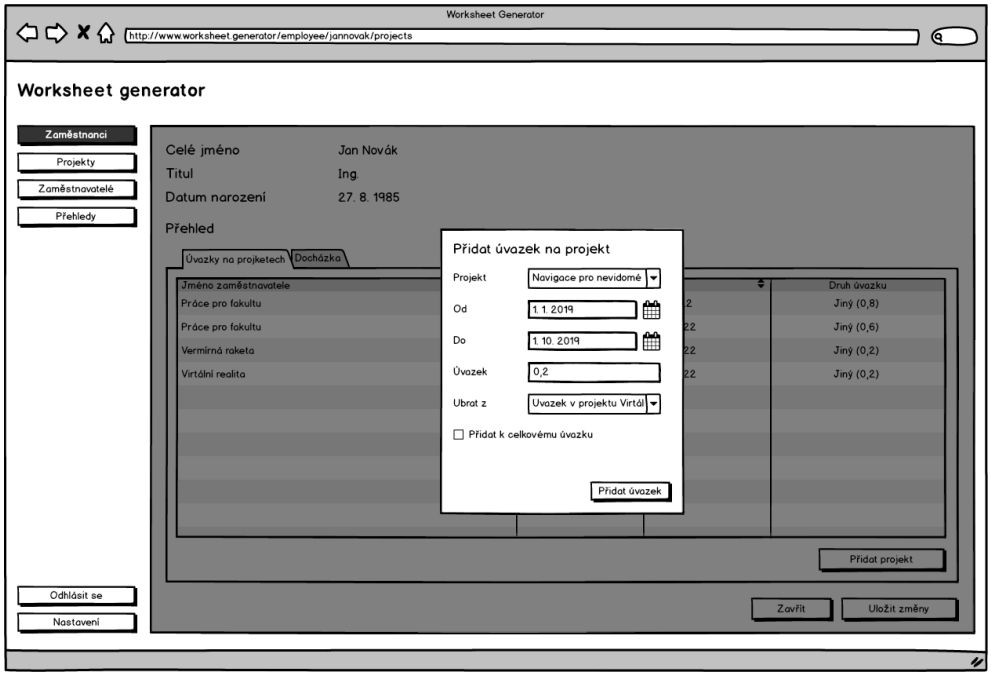
Obrázek G.3. Formulář přidání nového pracovního záznamu k zaměstnanci.



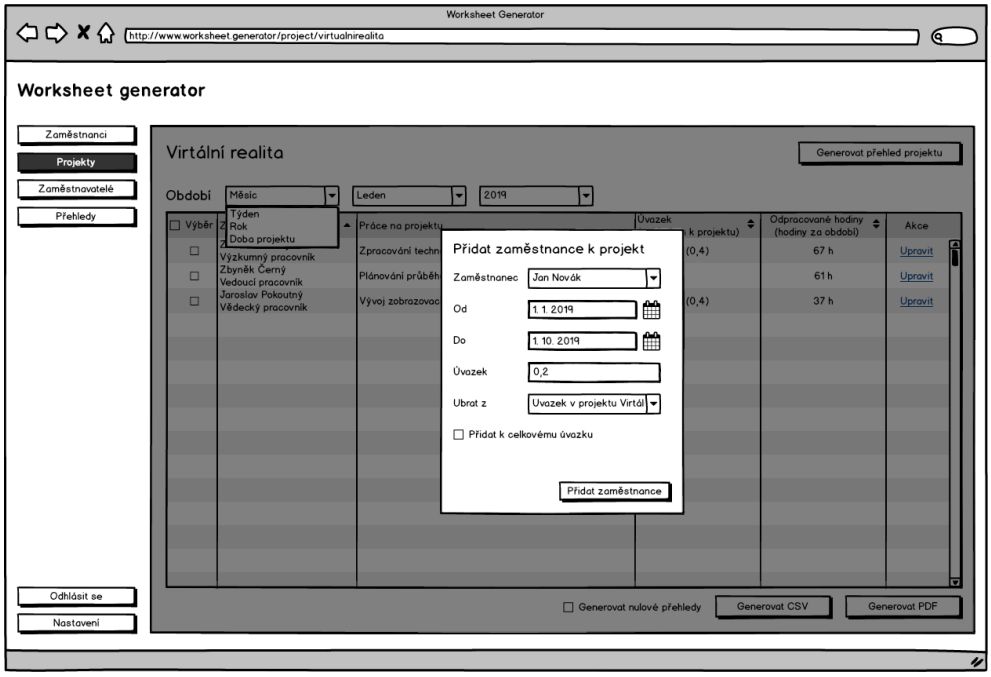
Obrázek G.4. Formulář úpravy pracovního záznamu u zaměstnance.



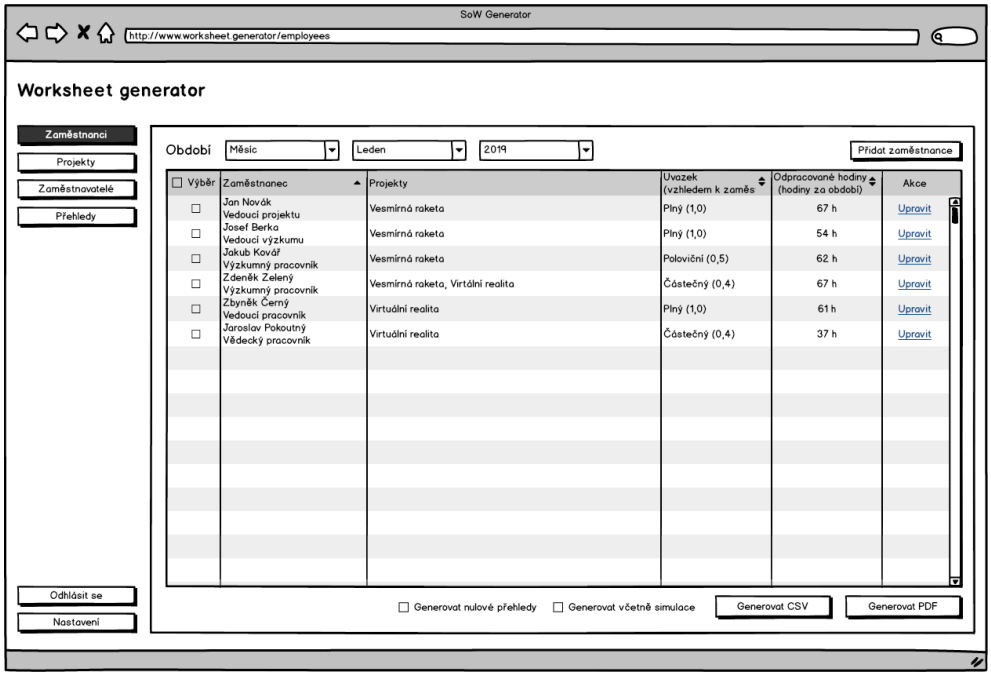
Obrázek G.5. Přehled projektů zaměstnance.



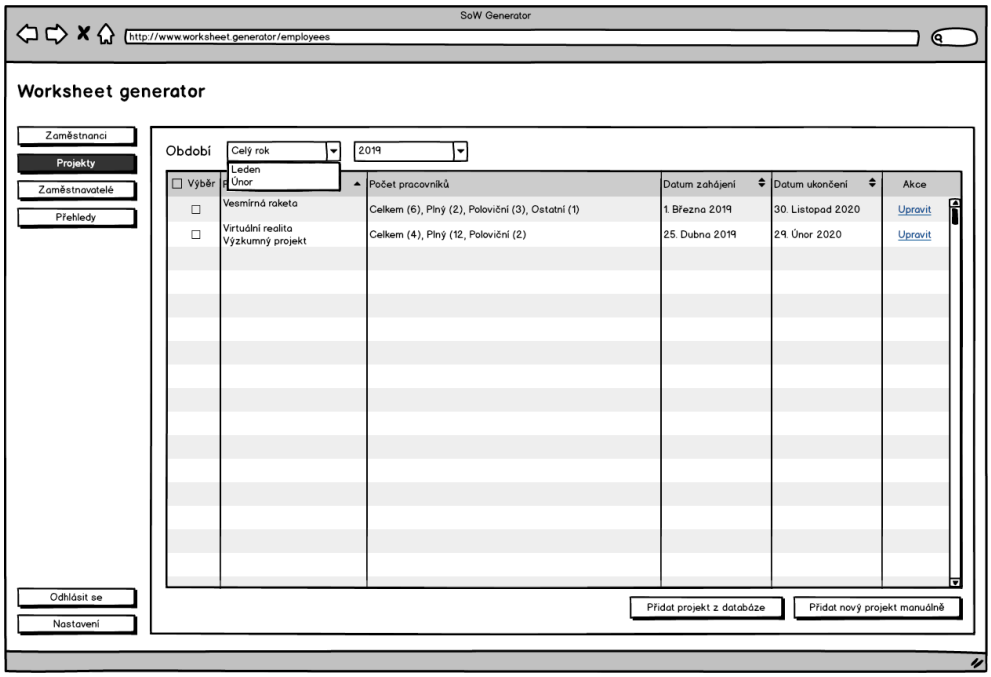
Obrázek G.6. Přidání úvazku na projektu k zaměstnanci.



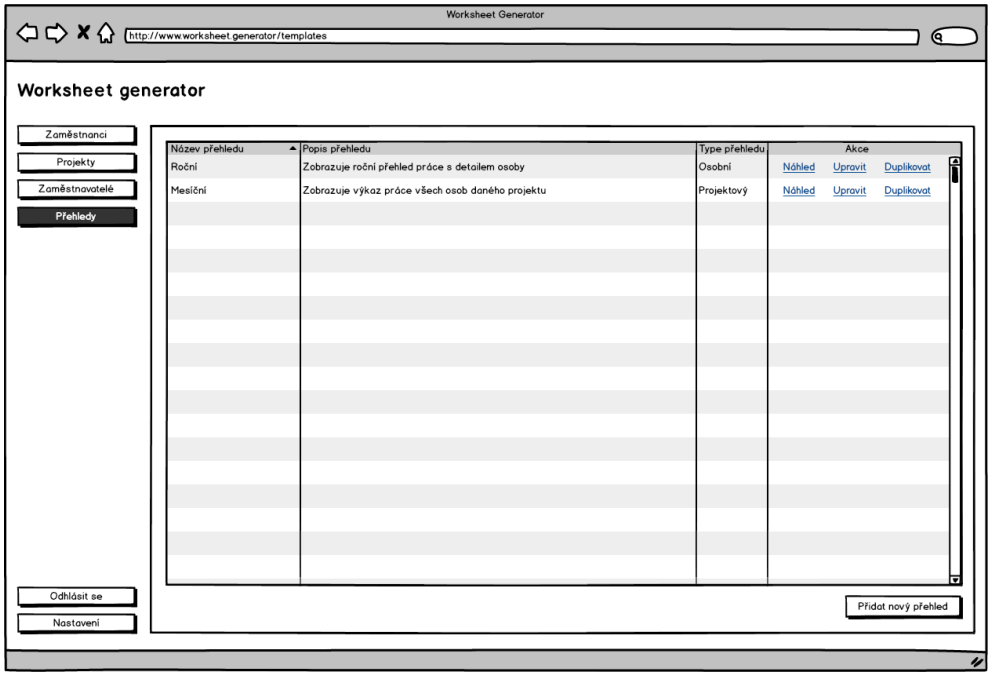
Obrázek G.7. Přidání zaměstnaneckého úvazku k projektu.



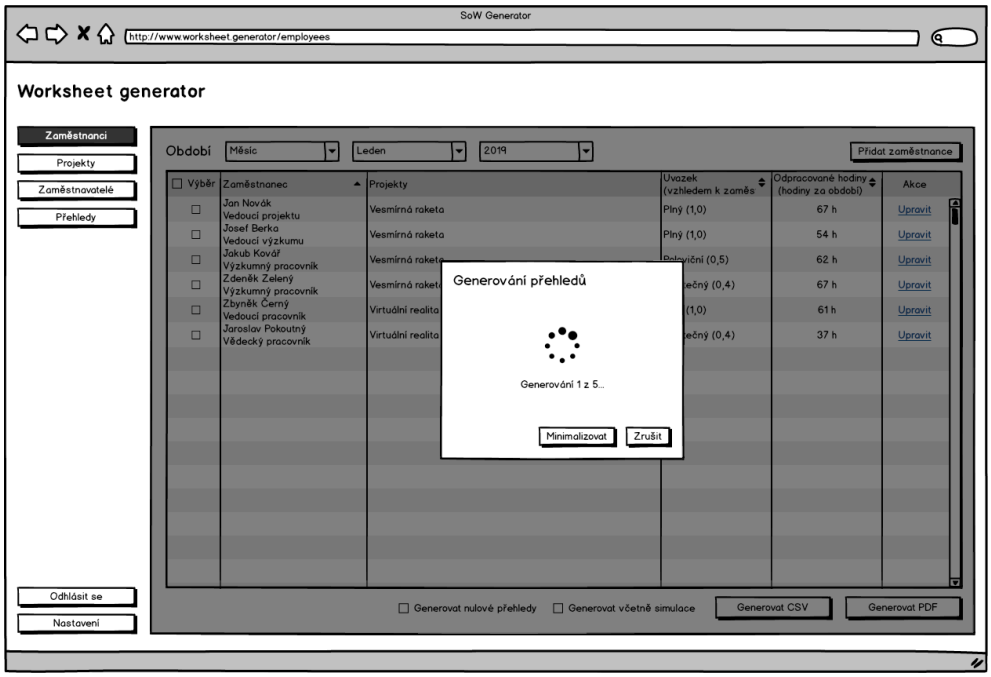
Obrázek G.8. Přehled všech zaměstnanců.



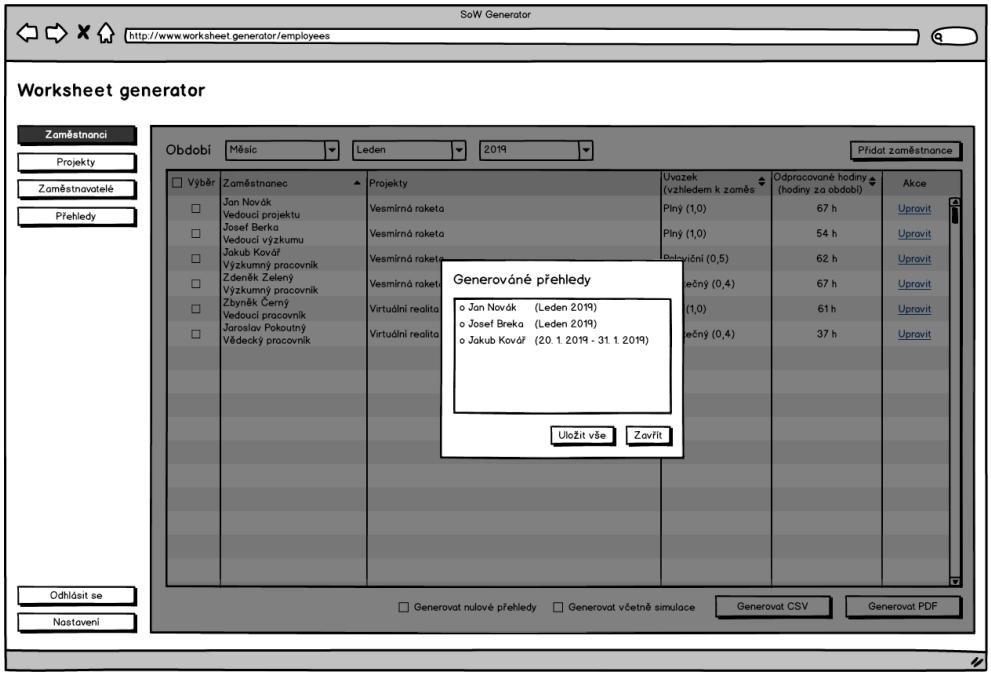
Obrázek G.9. Přehled všech projektů.



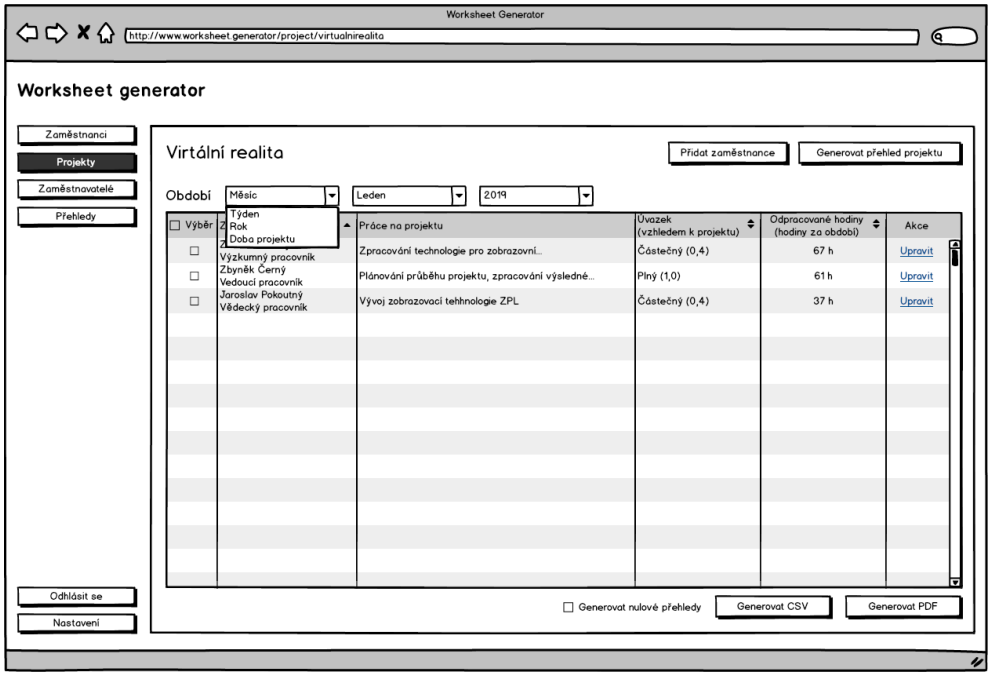
Obrázek G.10. Přehled všech šablon.



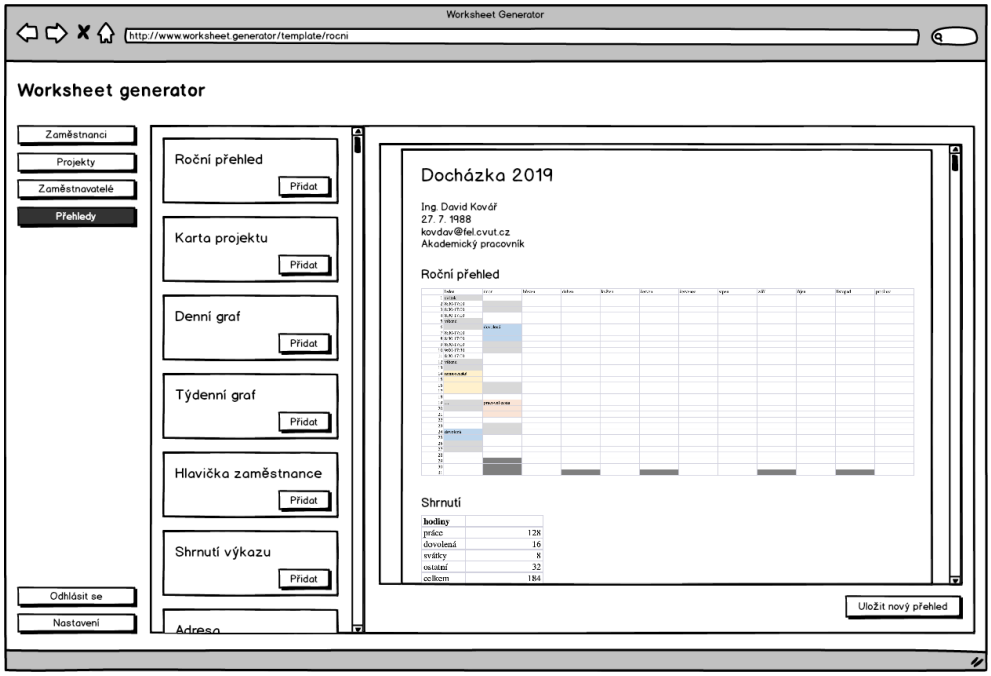
Obrázek G.11. Generování přehledů docházky.



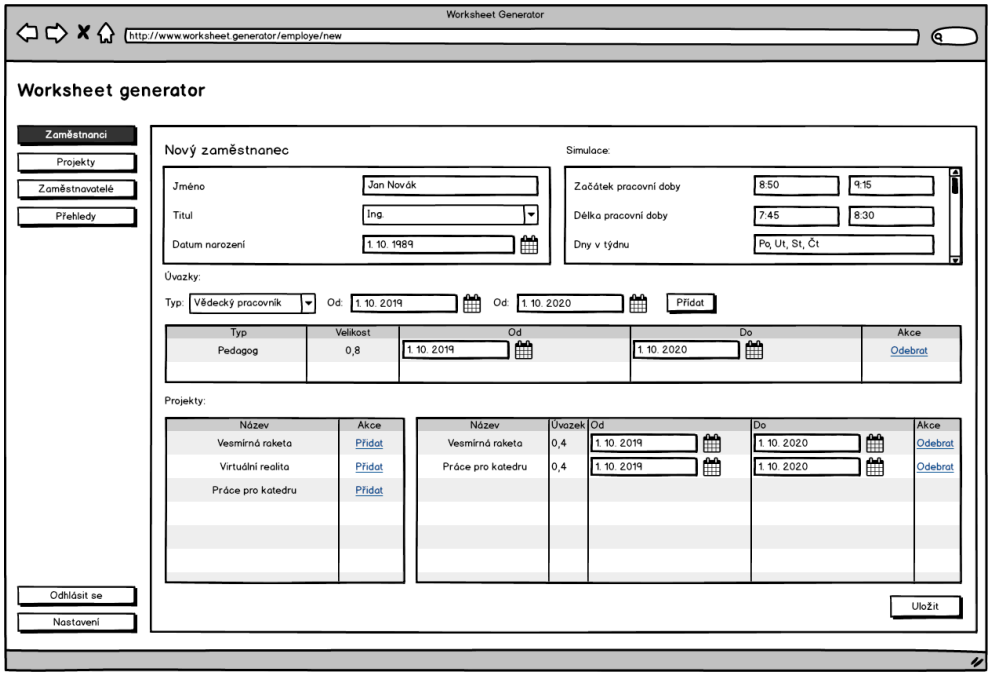
Obrázek G.12. Uložení vygenerovaných přehledů docházky.



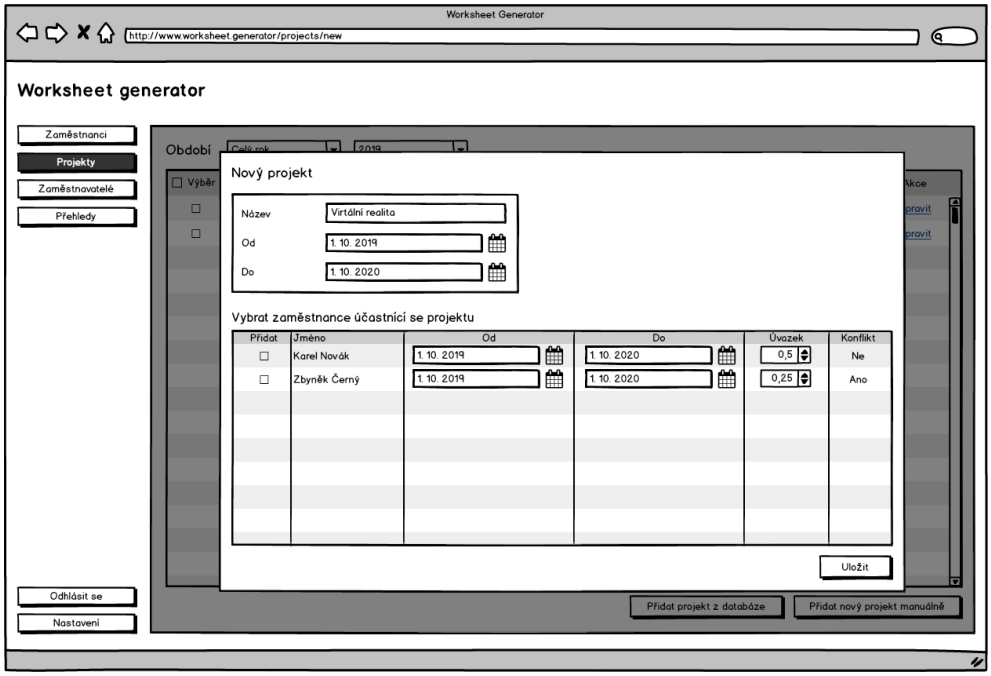
Obrázek G.13. Přehled projektu.



Obrázek G.14. Vytvoření nové šablony přehledu docházky.



Obrázek G.15. Přidání nového zaměstnance.



Obrázek G.16. Přidání nového projektu se zaměstnanci.