Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Docházkový systém

Bakalářská práce

Autor: Stanislav Čapek
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Hradec Králové duben 2021

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 27.4.2021 Stanislav Čapek

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Mgr. Tomášovi Kozlovi, Ph.D. za metodické vedení práce a za rady při zpracování této práce

Anotace

Tato práce si klade za cíl navrhnout a implementovat s využitím aktuálních nástrojů aplikaci Docházkový systém a popsat základní principy použitých technologií.

Na začátku práce se nachází kapitola Analýza a návrh aplikace, která zpracovává tematicky pohled na platnou legislativu a typ úkolů pro systém.

V rámci analýzy nejnovějších technologií jsou porovnány tři technologie pro tvorbu frontendu a tři technologie pro tvorbu backendu. Z těchto kandidátů je vybrána jedna technologie pro každou část.

Navržená aplikace je rozdělena na dvě samostatné části, frontend a backend, s důrazem na co nejmenší provázání. Frontend je navržen jako Single Page Application s využitím technologie React a dalších podpůrných knihoven. Backend je vytvořen jako REST API s využitím projektů ze Spring frameworku zejména Spring Data REST.

Annotation

Title: Attendance system

This work aims to design and implement the application Attendance System using current tools and describe the basic principles of technologies used.

At the beginning of the work is the chapter Analysis and design of the application that handles the applicable legislation and the type of tasks for the system.

As part of the analysis of the latest technologies, three technologies for creating a frontend and three for creating a backend are compared. From these candidates one technology is selected for each part.

The proposed application is divided into two separate parts, frontend and backend with an emphasis on low coupling. The frontend is designed as Single Page Application using React technology and other support libraries. Backend is formed as a REST API using the Spring Framework project especially Spring Data REST.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Tato práce se bude zabývat návrhem a řešením Docházkového systému s využitím aktuálních open source nástrojů.

Téma Docházkového systému bylo vybráno na základě předchozí znalosti problematiky evidování pracovní docházky v zaměstnání. Prvním krokem bylo plánování směn na kalendářní měsíc evidovaný do šablony plánu směn. Na začátku každého měsíce se poté zpracovala pracovní docházka za minulý měsíc, která následně sloužila pro mzdové oddělení jako podklad pro výpočet mzdy či platu. Vše se provádělo v rámci kancelářského softwaru MS Excel. Ten sám o sobě klade na uživatele určitou míru počítačové gramotnosti a pro některé méně zdatné pracovníky to představovalo jisté úskalí.

Navrhovaný systém si klade za cíl tato úskalí odstranit nebo alespoň minimalizovat. Z praxe se ukazuje, že MS Excel i přes svou možnou komplikovanost je dobrý nástroj pro jednoduché evidování směn. Tabulku lze graficky upravit, provádět jednoduché mezivýpočty apod. Následné převedení odpracovaných hodin, rozlišení příplatkových hodin a dovolených je již těžším úkolem. Proto systém bude navržen s přihlédnutím na maximální jednoduchost uživatelského rozhraní, které bude mít formu webové stránky. Způsob prohlížení webového obsahuje, resp. chování uživatele v rámci webové stránky, je již tak ustálené, že i pro méně počítačově gramotného uživatele bude obsluha aplikace přirozená. Systém by měl dovolovat vygenerovat, načíst a zpracovávat šablonu pro plánování směn. Po editaci pracovní docházky přes jednoduché rozhraní by měl systém být schopný vygenerovat podklad pro výpočet mzdy či platu.

Tímto způsobem by byla zajištěna zpětná kompatibilita se zvyklostmi a procesy evidování směn spolu s vypracováním pracovní docházky za využití automatických výpočtů hodin a kategorizování směn.

2 Problematika docházkových systémů

Téměř každý zaměstnanec, se setkal s určitým druhem docházkového systému. Docházkový systém může mít analogovou podobu, kdy zaměstnanci zaznamenávají do knihy příchody do a odchody ze zaměstnání. V dřívějších dobách byly docházkové systémy řešeny píchacími hodinami (lidově označovanými jako píchačky). Ty zaznamenávaly na kusu papíru příchody a odchody jeho proděravěním či později tiskem aktuálního času.

V aktuální době již technologie razantně postoupila kupředu. Příchodem zařízení z oblasti Internet of Things a jiných chytrých zařízení jsou píchací hodiny spíše raritou. Fyzická zařízení jako taková zůstávají, mění se ale jejich podoba a technologie. Na základě průzkumu dostupných docházkových systémů lze obecně konstatovat, že ve většině případů se jedná o stejný model. Na straně zákazníka je namontované fyzické zařízení, u kterého zaměstnanec zaznamenává příchod či odchod ze zaměstnání. Tato data jsou přenesena do systému, kde je možné je následně upravovat. Konkrétní provedení se již liší výrobce od výrobce. Zejména u novějších systémů lze pozorovat respektování nových trendů jak po hardwarové, tak i v softwarové stránce. Jsou využívány bezkontaktní čtečky či biometrická zařízení. V softwarové oblasti je znatelný trend nasazení aplikace v cloudovém prostředí, které umožňuje být nezávislý na hardwarové infrastruktuře. Toto řešení je výhodné zejména pro menší a střední firmy.

Následuje výběr několika společností, které nabízejí řešení docházkového systému a stručný popis jejich řešení.

Docházkový systém od firmy Aktion (1) nabízí komplexní řešení pro firmy všech velikostí. Jejich systém například dovoluje plánování směn u vícesměnného a nepřetržitého provozu, definice vztahů a nadřízených, hlídání přesčasů a doby odpočinku a jiné. Nabízí možnost bezklíčového řešení pomocí biometrického snímače otisku prstu. Produkt je možné provozovat na pronajatém cloudu či na serveru zákazníka. Možné využití docházky pro mzdové systémy.

Docházkový systém od firmy Alveno (2) nabízí taktéž komplexní řešení pro všechny velikosti firem. Svůj produkt nabízí ve formě cloudového řešení s neomezeným počtem přístupů i uživatelských licencí. Možné využití docházky pro mzdové systémy. V nabídce nejsou zahrnuty čtečky. Společnost je ale dodává a lze tedy předpokládat jejich propojení s jimi nabízeným docházkovým systémem.

Docházkový systém od firmy Ima (3) se velmi podobá systémům předešlých dvou firem. Jedná se o cloudové řešení s možností napojení na přístupové či mzdové systémy.

Firma Ron (4) nabízí velmi podobné řešení jako výše uvedené firmy. Cloudové řešení, možnost připojení přes různá uživatelská rozhraní. Možnost napojení na přístupové systémy a propojení s jinými agendami.

Řešení docházkového systému od firmy Z-Ware (5) se již liší od předešlých produktů. Nabízený produkt je primárně určen pro provozování na systému Windows. Výrobce uvádí možnost spravovat docházku přes internetový prohlížeč. Cloudové řešení výslovně neuvádí. Celkový dojem působí zastaralým a nepříliš inovativním způsobem.

V době psaní této práce probíhá třetí vlna pandemie viru COVID-19, který změnil způsob práce zaměstnanců. Nařízením vlády, ale i vlastní iniciativou zaměstnavatelů pracují zaměstnanci z domova. Práce z domova má svá specifika oproti práci v sídle zaměstnavatele. Je téměř nemožné kontrolovat zaměstnance fyzicky. Zaměstnavatel může například využít docházkového systému ke kontrole zaměstnance, kdy tento na začátku a na konci směny provede zápis do systému. Záznam je opatřen časovým razítkem jeho vytvoření.

3 Analýza a návrh aplikace

3.1 Legislativní rámec

Tato kapitola si klade za cíl poskytnout základní vhled do právní úpravy, jež se váže k tvorbě docházkového systému. Zároveň je však třeba podotknout, že při tvorbě docházkového systému je nutné se důkladně seznámit s prostředím, pro které je tvořen. Zákonná úprava obsahuje četné výjimky, které je třeba zohlednit. Stejně tak je třeba zohlednit případné další podmínky, které může zaměstnavatel zaměstnancům poskytovat (sick day, dovolená, home office, apod.) nad rámec zákona.

Základní právní úpravu nalezneme v zákoně[1] č. 262/2006 Sb., zákoník práce, ve znění pozdějších předpisů (dále jen „zákon“). Tento zákon upravuje právní vztahy vznikající při výkonu závislé práce mezi zaměstnanci a zaměstnavateli, tyto vztahy jsou vztahy pracovněprávními. Smysl a účel ustanovení tohoto zákona vyjadřují i základní zásady pracovněprávních vztahů, uvedené v § 1a zákona. Jsou jimi zejména:

„a) zvláštní zákonná ochrana postavení zaměstnance,

b) uspokojivé a bezpečné podmínky pro výkon práce,

c) spravedlivé odměňování zaměstnance,

d) řádný výkon práce zaměstnancem v souladu s oprávněnými zájmy zaměstnavatele,

e) rovné zacházení se zaměstnanci a zákaz jejich diskriminace.

Zásady zvláštní zákonné ochrany postavení zaměstnance, uspokojivých a bezpečných pracovních podmínek pro výkon práce, spravedlivého odměňování zaměstnance, rovného zacházení se zaměstnanci a zákazu jejich diskriminace vyjadřují hodnoty, které chrání veřejný pořádek.“

[1] Stav k 15.1.2021

Zákon v § 78 a násl. definuje základní pojmy, jejichž znalost je nutná při tvorbě docházkového systému. Jedná se o:

„a) pracovní doba je doba, v níž je zaměstnanec povinen vykonávat pro zaměstnavatele práci, a doba, v níž je zaměstnanec na pracovišti připraven k výkonu práce podle pokynů zaměstnavatele,

b) dobou odpočinku je doba, která není pracovní dobou,

c) směnou je část týdenní pracovní doby bez práce přesčas, kterou je zaměstnanec povinen na základě předem stanoveného rozvrhu pracovních směn odpracovat,

d) dvousměnným pracovním režimem je režim práce, v němž se zaměstnanci vzájemně pravidelně střídají ve 2 směnách v rámci 24 hodin po sobě jdoucích,

e) vícesměnným pracovním režimem je režim práce, v němž se zaměstnanci vzájemně pravidelně střídají ve 3 nebo více směnách v rámci 24 hodin po sobě jdoucích,

f) nepřetržitým pracovním režimem je režim práce, v němž se zaměstnanci vzájemně pravidelně střídají ve směnách v nepřetržitém provozu zaměstnavatele v rámci 24 hodin po sobě jdoucích,

g) nepřetržitým provozem provoz, který vyžaduje výkon práce 24 hodin denně po 7 dnů v týdnu,

h) pracovní pohotovostí doba, v níž je zaměstnanec připraven k případnému výkonu práce podle pracovní smlouvy, která musí být v případě naléhavé potřeby vykonána nad rámec jeho rozvrhu pracovních směn. Pracovní pohotovost může být jen na jiném místě dohodnutém se zaměstnancem, odlišném od pracovišť zaměstnavatele,

i) prací přesčas se rozumí práce konaná zaměstnancem na příkaz zaměstnavatele nebo s jeho souhlasem nad stanovenou týdenní pracovní dobu vyplývající z předem stanoveného rozvržení pracovní doby a konaná mimo rámec rozvrhu pracovních směn. U zaměstnanců s kratší pracovní dobou je prací přesčas práce přesahující stanovenou týdenní pracovní dobu; těmto zaměstnancům není možné práci přesčas nařídit. Prací přesčas není, napracovává-li zaměstnanec prací konanou nad stanovenou týdenní pracovní dobu pracovní volno, které mu zaměstnavatel poskytl na jeho žádost,

j) noční prací práce konaná v noční době; noční doba je doba mezi 22. a 6. hodinou,

k) zaměstnancem pracujícím v noci je zaměstnanec, který odpracuje během noční doby nejméně 3 hodiny ze své pracovní doby v rámci 24 hodin po sobě jdoucích v průměru alespoň jednou týdně v období, uvedeném v § 94 odst. 1 zákona[2],

l) rovnoměrným rozvržením pracovní doby rozvržení, při kterém zaměstnavatel rozvrhuje na jednotlivé týdny stanovenou týdenní pracovní dobu, popřípadě kratší pracovní dobu,

m) nerovnoměrným rozvržením pracovní doby rozvržení, při kterém zaměstnavatel nerozvrhuje rovnoměrně na jednotlivé týdny stanovenou týdenní pracovní dobu, popřípadě kratší pracovní dobu, s tím, že průměrná týdenní pracovní doba nesmí přesáhnout stanovenou týdenní pracovní dobu, popřípadě kratší pracovní dobu, za období nejvýše 26 týdnů po sobě jdoucích. Jen kolektivní smlouva může toto období vymezit nejvýše na 52 týdnů po sobě jdoucích. „

Zákon výslovně v § 81 stanoví, že „*pracovní dobu rozvrhuje zaměstnavatel, který také určuje určí začátek a konec směn. Pracovní doba se rozvrhuje zpravidla do pětidenního pracovního týdne. Při rozvržení pracovní doby je zaměstnavatel povinen přihlédnout k tomu, aby toto rozvržení nebylo v rozporu s hledisky bezpečné a zdraví neohrožující práce*[3].“ Povinností zaměstnance je být na začátku směny na svém pracovišti a odcházet z něho až po skončení směny, jejíž délka nesmí přesáhnout 12 hodin.

Zákon upravuje v § 79 stanovenou týdenní pracovní dobu, která činí 40 hodin týdně, přičemž zákon stanoví i výjimky u specifických prací (práce v podzemí, vícesměnný režim, apod.).

Výše uvedené je názorným příkladem toho, jak je třeba znát konkrétní podmínky daného zaměstnavatele, pro nějž je docházkový systém tvořen.

---

[2] Délka směny zaměstnance pracujícího v noci nesmí překročit 8 hodin v rámci 24 hodin po sobě jdoucích; není-li to z provozních důvodů možné, je zaměstnavatel povinen rozvrhnout stanovenou týdenní pracovní dobu tak, aby průměrná délka směny nepřekročila 8 hodin v období nejdéle 26 týdnů po sobě jdoucích, přičemž při výpočtu průměrné délky směny zaměstnance pracujícího v noci se vychází z pětidenního pracovního týdne.

[3] Zaměstnavatel je např. povinen poskytovat přestávky v práci a bezpečnostní přestávky za podmínek uvedených v § 88 a § 89 zákona

Z pohledu docházkového systému je nejpodstatnějším ustanovením zákona jeho § 96, který říká, že „*zaměstnavatel je povinen vést u jednotlivých zaměstnanců evidenci s vyznačením začátku a konce odpracované směny, práce přesčas, noční práce, doby v době pracovní pohotovosti a pracovní pohotovosti, kterou zaměstnanec držel. Na žádost zaměstnance je zaměstnavatel povinen umožnit zaměstnanci nahlédnout do jeho účtu pracovní doby nebo evidence pracovní doby a do jeho účtu mzdy a pořizovat si z nich výpisy, popřípadě stejnopisy na náklady zaměstnavatele*.“

Zaměstnavatel může pracovní dobu rozvrhovat několika způsoby. Zvláštním způsobem rozvržením pracovní doby je pružné rozvržení pracovní doby, uvedené v § 85 zákona, a konto pracovní doby uvedené v § 86 zákona. „*Pružné rozvržení pracovní doby zahrnuje časové úseky základní a volitelné pracovní doby, jejichž začátek a konec určuje zaměstnavatel*. *Konto pracovní doby je způsob rozvržení pracovní doby, který smí zavést jen kolektivní smlouva nebo vnitřní předpis u zaměstnavatele, u kterého nepůsobí odborová organizace a nesmí být uplatněno u zaměstnavatelů uvedených v § 109 odst. 3*[4].“

Pro účely této práce je třeba zmínit i překážky v práci, které se sice za výkon práce nepovažují, ale je nutné je v docházkovém systému evidovat. Překážky v práci se dělí do dvou skupin – překážky na straně zaměstnance a překážky na straně zaměstnavatele. Překážky v práci na straně zaměstnance jsou uvedeny v § 191 a násl. zákona. Jedná se o *důležité osobní překážky* (pracovní neschopnost, karanténa, mateřská nebo rodičovská dovolené a další) a *jiné důležité osobní překážky v práci*

[4] Zaměstnavatelem, kterým je stát, územní samosprávný celek, státní fond, příspěvková organizace, jejíž náklady na platy a odměny za pracovní pohotovost jsou plně zabezpečovány z příspěvku na provoz poskytovaného z rozpočtu zřizovatele nebo z úhrad podle zvláštních právních předpisů, školská právnická osoba zřízená Ministerstvem školství, mládeže a tělovýchovy, krajem, obcí nebo dobrovolným svazkem obcí podle školského zákona, nebo regionální rada regionu soudržnosti, s výjimkou peněžitého plnění poskytovaného občanům cizích států s místem výkonu práce mimo území České republiky.

(svatba, pohřeb, návštěva lékaře, apod.), které stanoví vláda nařízením[5]. Poslední kategorií, pak jsou překážky v práci z důvodu obecného zájmu, uvedené v § 200 a násl. zákona (výkon veřejné funkce, výkon občanské povinnosti, apod.). Překážky na straně zaměstnavatele jsou upraveny § 207 a násl. zákona, a jedná se o prostoje a přerušení práce způsobené nepříznivými povětrnostními vlivy a o jiné překážky v práci na straně zaměstnavatele (dočasné omezení odbytu výrobků zaměstnavatele).

Závěrem jen krátce zmínka o evidenci dovolené, která je od 1.1.2021 počítána v hodinách. Zákon v § 211 rozlišuje 3 typy dovolené:

„a) dovolená za kalendářní rok

b) poměrnou část dovolené za kalendářní rok

c) dodatková dovolená.“

Výměra dovolené činí nejméně 4 týdny v kalendářním roce a výměra dovolené zaměstnanců zaměstnavatelů uvedených v § 109 odst. 3[6] činí 5 týdnů v kalendářním roce. Výměra dovolené pedagogických pracovníků a akademických pracovníků vysokých škol činí 8 týdnů v kalendářním roce.

---

[5] Nařízení vlády č. 590/2006 Sb., kterým se stanoví okruh a rozsah jiných důležitých osobních překážek v práci

[6] Viz poznámka č. 4

3.2 Analýza požadavků na docházkový systém

Doménový problém je definován následovně: systém je tvořen pro zaměstnavatele, který je definován v § 109 odst. 3 zákona, přičemž se jedná pouze o jeho část - obecní policii. Tato má specifické požadavky na vlastnosti docházkového systému, spočívající v zaznamenání těchto událostí:

- evidence směn
- odchod a příchod ze zaměstnání
- přesčasy
- sick day
- překážky v práci na straně zaměstnance
- dovolená
- odpočinek mezi směnami
- přestávka v práci
- nerovnoměrné rozvržení pracovní doby
- státní svátek
- pracovní cesta
- odpracované hodiny
- neodpracované hodiny

Další – zejména funkční – požadavky zaměstnavatele na docházkový systém jsou tyto:

- každý uživatel má své vlastní přístupy, pod kterými bude zadávat jednotlivé události. Každý kalendářní měsíc lze zobrazit samostatně,
- každý uživatel bude mít přiřazen uživatelský účet,
- uživatelskému účtu bude přiřazena role, a to minimálně jedna, na základě role či rolí je uživateli umožněno provádět určité akce,
- upravenou evidenci je možné vytisknout nebo uložit do souboru,
- automaticky bude přednastavena pracovní doba od 7 do 19 v případě denní směny a od 19 do 7 v případě noční směny,

- docházkový systém bude automaticky reflektovat kalendář, tedy rozpozná dny, na které připadne státní svátek a dny pracovního klidu,
- vyrovnávací období činí 26 týdnů,
- fond pracovní doby je vypočten na měsíce dle stanovené týdenní pracovní doby, přičemž týdenní pracovní doba je stanovena na 37,5 hodiny týdně, případně kratší pracovní doba dle § 80 zákona,
- směny jsou plánovány v excelové tabulce dle zadané šablony, šablonu je možné vygenerovat ze systému pro všechny uživatele a bude zahrnovat celý kalendářní rok,
- po vložení šablony do docházkového systému dojde k automatickému předvyplnění pracovní docházky,
- vedoucí pracovník bude mít možnost uzavřít kalendářní měsíc a tento bude později přístupný pouze pro náhled,

3.2.1 Příklad z praxe

Typová úloha je příkladem, který může odpovídat příkladu z praxe. A jedná se o takový typ úlohy, který navrhovaný docházkový systém má za úkol zpracovat. Strážník obecní policie je zaměstnanec, který má stanovenou týdenní pracovní dobu na 37,5 hodiny týdně a pracuje v nepřetržitém pracovním režimu. Zaměstnavatel plánuje pracovní dobu nerovnoměrně vždy na následující kalendářní měsíc.

Směny zaměstnanci byly naplánované a odpracované v měsíci července roku 2021. Z předešlého měsíce jsou zaměstnanci převedeny 3 odpracované hodiny nad rámec pracovního fondu. Vzhledem k tomu, že se jedná o nepřetržitý pracovní režim práce, jsou mezi pracovní dny započítány i dny na které připadne státní svátek. V měsíc červenec roku 2021 připadne 22 pracovních dnů a z toho je vypočteno, že v daném měsíci činí fond pracovní doby 165 hodin. V následující tabulce je přehled datumů a druh odpracovaných směn.

Tabulka 1 Přehled naplánovaných a odpracovaných směn

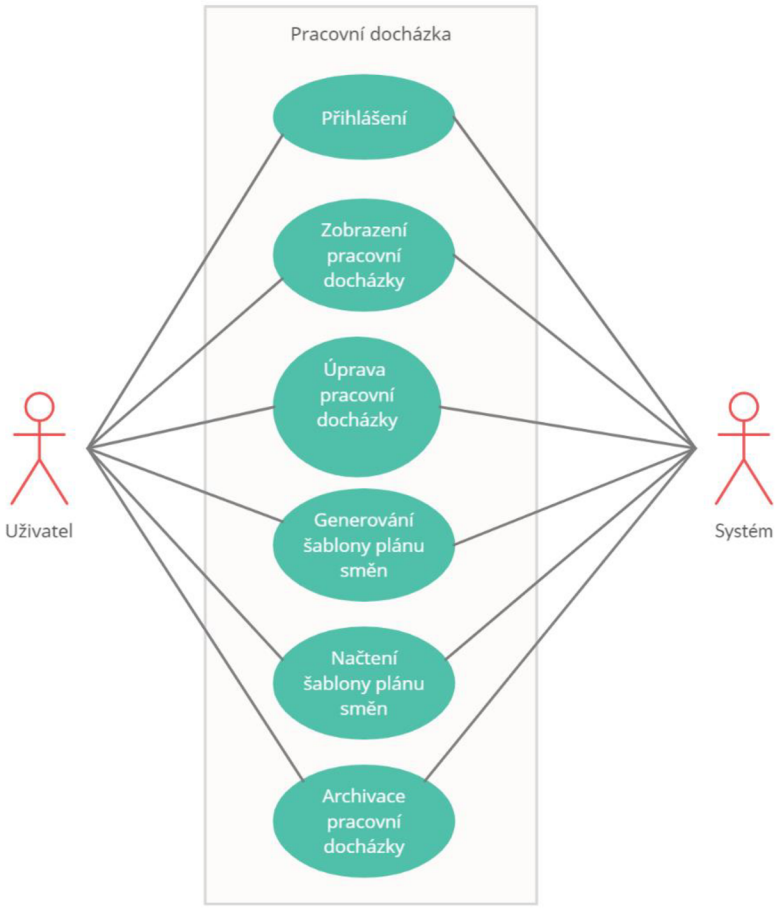
Datum	Druh	délka v hod.
01.07.2021	denní	12
02.07.2021	noční	12
06.07.2021	denní	12
07.07.2021	denní	12
08.07.2021	zdravotní volno	12
13.07.2021	denní	12
14.07.2021	denní	12
17.07.2021	noční	12
20.07.2021	řádná dovolená	12
21.07.2021	denní	12
22.07.2021	denní	12
26.07.2021	denní	12
27.07.2021	denní	12
30.07.2021	noční	12

Zdroj: vlastní zpracování

Zaměstnanci se z předchozího měsíce převádí 3 odpracované hodiny. Pohledem na evidované hodiny bez hlubší analýzy, zda se například jedná o řádně odpracované hodiny, zjistíme, že zaměstnanci se započítá 168 hodin.

Celkové odpracované hodiny se vypočítají sečtením odpracovaných hodin a hodin převedených z měsíce předešlého. Tedy $168 + 3 = 171$ odpracovaných hodin. Od odpracovaných hodin je odečten fond pracovních hodin na daný měsíc, $171 - 165 = 6$ hodin, které se převádějí do dalšího měsíce.

3.3 Typové úlohy



Obr. 1 UseCase diagram

Zdroj: vlastní zpracování

Tato část práce je zaměřena na návrh systému pohledem typových úloh. Typové úlohy jsou rozděleny na funkční a non-funkční požadavky. Funkční požadavky a non-funkční požadavky jsou rozděleny, protože se jedná o koncepčně jiné požadavky na navrhovaný systém. *„Funkční požadavky vyjádřené v uživatelských příbězích, funkcích a schopnostech představují většinu práce na vytváření řešení, která*

mají pro uživatele hodnotu."[7] Scaled Agile, Inc. (6) (překlad autora). *„Nefunkční požadavky ... definují systémové atributy, jako je bezpečnost, spolehlivost, výkon, udržovatelnost, škálovatelnost a použitelnost. Slouží jako omezení nebo omezení návrhu systému napříč různými nevyřízenými položkami."*[8] Scaled Agile, Inc. (6) (překlad autora). Jedná se rozpracování požadavků uvedených úvodu kapitoly 3.2.

3.3.1 Přihlášení uživatele

Funkční požadavek: Uživatelem může být osoba nebo systém. Uživateli je vytvořený uživatelský přístup s přiřazenou rolí nebo rolemi. Uživateli je minimálně jedna role přiřazena. Role slouží k autorizaci provedení úkonů v rámci systému. Uživatelským přístupem je myšleno uživatelské jméno a heslo, kterým se uživatel autentizuje v systému.

Pro typovou úlohu Uživatelský přístup vyplývají požadavky na návrh uživatelského rozhraní a serverové části.

Uživatel je schopen v uživatelském rozhraní zadat své uživatelské jméno a heslo. V případě zadání špatné kombinace jména a hesla je uživatel na toto upozorněn. Proběhne-li autentizace v pořádku, je uživatel přesměrován na správu své docházky v aktuálním měsíci.

Non-funkční požadavek: Přenos mezi uživatelským rozhraním a serverem je zabezpečen takovým způsobem, aby nedošlo k prozrazení uživatelských přístupových údajů.

---

[7] Functional requirements expressed in user stories, features, and capabilities represent most of the work in building solutions that deliver value to the user.

[8] Nonfunctional Requirements ... define system attributes such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across the different backlogs.

3.3.2 Zobrazení pracovní docházky

Funkčním požadavkem pro zobrazení pracovní docházky je, že uživateli je umožněno prohlížet docházku zpětně s následujícím omezením. Docházka byla založena/vytvořena v měsíci, pro který je vedena, anebo v měsíci následujícím. Uživatel může zobrazit zpětně všechny pracovní docházky, které splňují předchozí omezení. Uživatel může zobrazit, případně založit, docházku pro následující měsíc měsíce aktuálního. Na zobrazení docházky je autorizován.

Typová úloha zobrazení pracovní docházky nemá specifické non-funkční požadavky.

3.3.3 Úprava pracovní docházky

Funkční požadavky: Úprava pracovní docházky podléhá následujícím omezením. Uživatel je pro úpravu pracovní docházky autorizován. Pracovní docházka se nachází ve stavu odemčeno. Pracovní docházky lze upravovat pouze měsíc zpětně a s měsíčním předstihem od aktuálního měsíce.

Pracovní docházka se skládá ze všech dnů daného měsíce. V jednotlivých dnech lze nastavit počátek a konec směny a typ směny. Pro vybrané typy směn jsou přednastaveny hodnoty začátku a konce směny.

Změny provedené v uživatelském rozhraní jsou synchronizovány se serverem. Ovlivní-li změna odpracované hodiny, projeví se tato změna i v přepočtu odpracovaných hodin v daném měsíci a převodu hodin do dalšího měsíce.

Vyrovnávací období je nastaveno ke konci měsíce červen. Hodiny, které jsou nad rámec, resp. převáděly by se do dalšího měsíce, jsou zaevidovány jako přesčasové hodiny.

Non-funkční požadavky: Rychlost odezvy serveru na provedenou změnu některých údajů docházky by neměla přesáhnout 3 vteřiny. Grafické zvýraznění dnů, na které připadne státní svátek.

3.3.4 Archivace pracovní docházky

Funkční požadavek: Každou pracovní docházku, kterou lze zobrazit, je možné vytisknout nebo uložit do souboru.

Mezi non-funkční požadavky můžeme zařadit omezení maximální doby trvání generování docházky pro účely tisku nebo uložení do souboru. Tato doba nesmí přesáhnout 60 vteřin.

3.3.5 Generování šablony plánu směn

Navrhovaný systém umožňuje vygenerovat soubor ve formátu XLSC, který slouží k plánování směn zaměstnanců na kalendářní rok. Soubor je generován podle předem dané šablony. Šablona obsahuje všechny zaměstnance.

Mezi non-funkční požadavky můžeme zařadit omezení maximální doby trvání generování docházky pro účely tisku nebo uložení do souboru. Tato doba nesmí přesáhnout 120 vteřin. Vygenerovaný soubor bude ve formátu XLSX.

3.3.6 Načtení vyplněné šablony plánu směn

Funkční požadavky: Navrhovaný systém umožňuje načíst vygenerovanou šablonu popsanou v 3.3.5. Z načtené šablony se extrahuje plán směn pro aktuální měsíc a měsíc předchozí, a to pouze pro aktuálně přihlášeného uživatele.

K uložení extrahovaných směn a zapsání do docházky z obou měsíců dojde za předpokladu, že jsou splněny tyto podmínky: měsíc se nachází v aktuálním roku a zároveň docházka pro daný měsíc buď neexistuje, nebo není označena jako zamčená.

Po úspěšném načtení je uživatelské rozhraní synchronizováno se serverem.

Mezi non-funkční požadavky můžeme zařadit omezení maximální doby trvání zpracování nahrané šablony. Tato doba nesmí přesáhnout 120 vteřin.

4 Výběr a popis technologií

4.1 Přehled výběru dostupných technologií pro tvorbu klienta

4.1.1 React

React je javascriptová knihovna určená pro tvorbu uživatelského rozhraní, která je zpravidla nasazena jako SPA (Single Page Application). Byla vyvíjena společností Facebook jako podniková knihovna. Změna nastala v roce 2012, kdy společnost Facebook odkoupila sociální síť Instagram. Jak řekl Tom Occhino na konferenci JSConfUS 2013 (7) právě tento akt převzetí byl spouštěcím motorem pro refaktorování knihovny React a doplnění dokumentace jako by šlo o open source projekt. Společnost Facebook Inc. úspěšně používala React na své sociální síti, a po úpravách knihovny byl Instagram první externí aplikací využívající knihovnu. V roce 2013 došlo k uvolnění knihovny React jako open source. V době psaní této práce přispívá do knihovny React přes platformu Github více jak 1500 lidí (8) a je využívána ve více jak 5 miliónech projektech (8).

Hlavní myšlenkou knihovny React je deklarování funkčních, upravitelných, znovupoužitelných elementů neboli komponent. Komponenty jsou uspořádány do stromové struktury. Data jsou předávána z předka na jeho potomky pomocí vlastností (props). Komponenta může nabývat stavu (state), pak je považována za stavovou komponentu, nebo neobsahuje stav a jedná se o bezstavovou komponentu. Bezstavová komponenta vrací pouze zobrazení v určitém časovém okamžiku. Stav jako takový je pak uchován v rodičovské komponentě. Tímto způsobem mohou komponenty, které jsou potomky rodičovské komponenty, sdílet společný stav.

Dále lze komponenty rozlišovat podle toho, zda se jedná o komponentu třídy nebo funkční komponentu (9). Komponenta třídy je deklarována klíčovým slovem class a rozšiřuje základní třídu React.Component. Funkční komponenta je deklarována klíčovým slovem function. V dřívějších verzích knihovny se stav mohl nacházet jen v komponentě třídy, to však neplatí s příchodem háčků (hooks), které mimo jiné dovolují funkčním komponentám mít i svůj vlastní stav. Tento přístup je dokonce

tvůrci knihovny doporučován (10). Existuje několik dalších háčků, včetně možnosti vytvoření vlastního. Prozkoumání této mechaniky je však již nad rámec tohoto textu.

Pojem SPA, neboli v překladu jednostránková aplikace, je použit několikrát v textu této práce, proto autor pokládá za důležité tento pojem vysvětlit. SPA je druhem architektury webové stránky, jejíž hlavním vstupní bodem je pouze jediný HTML soubor (index.html). David Flanagan (11) ve své knize JavaScript: The Definitive Guide souvislosti s SPA uvádí, že následné fungování aplikace záleží na jejím druhu. Existují aplikace, jako například hry spustitelné v prohlížeči, které uživateli nabídnou úvodní obrazovku, a jakmile stáhne celý obsah aplikace, nejsou třeba další dodatečná volání směrem na server. Druhý obvyklý typ aplikace naopak ze začátku načte pouze nutná data k zobrazení uživatelského rozhraní a tím, jak uživatel používá aplikaci, dochází k vytvoření dodatečných požadavků směrem na server. Tímto způsobem dochází k dynamické změně obsahu v aplikaci bez nutnosti načtení jiné stránky či obnovení stávající.

4.1.2 Angular

Angular je Typescript open source framework, který nabízí kompletní řešení při vývoji SPA a je vyvíjen a spravován společností Google. V prvé řadě je zapotřebí rozlišit mezi AngularJS a Angular. AngularJS je první verze tohoto frameworku využívající programovacího jazyku Javascript, jak už značí jeho název. Tato verze byla zveřejněna v roce 2010. Během následujících šesti let došlo k úpravám frameworku a jeho vydání pod novým názvem Angular, někdy také označovaném jako Angular 2+. Dále v textu bude použito pouze označení Angular a mínit se tím bude verze druhá až nejnovější (12).

Angular se řídí vzorem MVC, tedy Model-View-Controller. MVC vzor není nikterak závislý na programovacím jazyku a je hojně využíván jako architektura aplikace. Principem MVC je oddělit data (model) od zobrazení (view). Propojení těchto dvou entit zajišťuje právě kontrolér (controller). Angular využívá tento vzor na úrovni jednotlivých komponent. Komunikace mezi zobrazením a modelem probíhá pomocí dvousměrného mapování dat. Změna vyvolaná např. v datech se projeví v zobrazení

a naopak. Mapování dat může být i jednosměrné a vše záleží na konkrétní implementaci kontroléru. Pomocí techniky vkládání závislostí (dependency injection) lze využívat např. HTTP, Routing nebo Forms Angular modulů, které rozšíří komponenty o další funkcionality. Stejným způsobem můžeme využít vlastní služby (services) k výměně dat nebo funkcionalit mezi jednotlivými komponentami.

Jak popisuje oficiální dokumentace ke knihovně (13), komponenta může být složena z několika částí a do jisté míry představuje vzor MVC. Model představuje třída, která obsahuje data a případně přidružené chování. View je reprezentováno souborem html, který funguje jako šablona. Pomocí speciálních zástupných znaků a syntaxí jsou vložena data. Vytvoří se tak zobrazení odpovídající stavu dat v konkrétním čase.

4.1.3 Vue

„Vue (výslovnost /vju:/, obdobně jako anglické slovo view) je progresivní framework pro budování uživatelského rozhraní. Narozdíl od ostatních monolitických frameworků je Vue navržen od začátku tak, aby mohl být postupně osvojován. Jádro knihovny se soustředí pouze na zobrazovací vrstvu a je tak jednodušší jej integrovat s ostatními knihovnami nebo existujícími projekty. Na druhou stranu, Vue je perfektně schopné pohánět sofistikované jednostránkové aplikace, za předpokladu, že se spojí s moderními nástroji a podpůrnými knihovnami.“[9] (14) (překlad autora)

Vue framework je také zaměřen na komponenty. Uživatelské rozhraní je pak skládáno do stromové struktury z jednotlivých komponent. Autoři Vue se znatelně inspirovali knihovnou React a frameworkem Angular. Vznikla tak kombinace obou zmíněných knihoven.

---

[9]Vue (pronounced /vju:/, like view) is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.

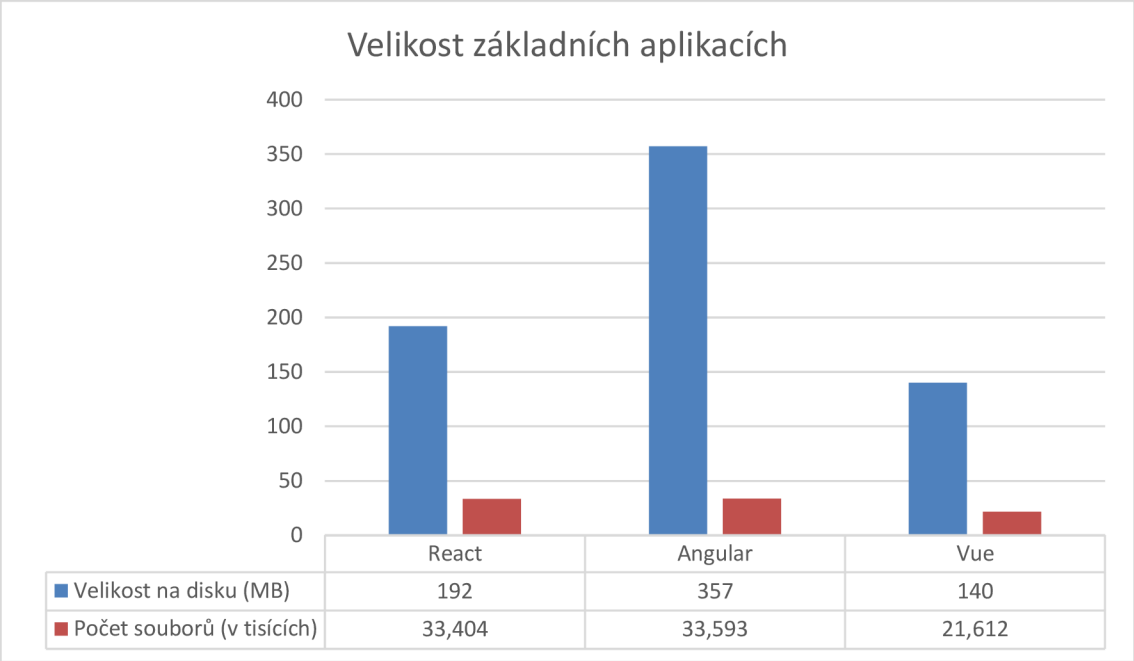
Podobnost struktury Vue komponenty s komponentou Angularu je zejména v separování do sekcí MVC. Na rozdíl od komponenty Angularu jsou jednotlivé části obsaženy v jednotném souboru s vlastní příponou .vue. Podle uživatelské dokumentace knihovny Vue (15) ve verzi 3 zapouzdření šablony, logiky a stylu do jedné komponenty vytváří prostředí vhodné pro oddělení odpovědnosti mezi jednotlivými komponentami.

4.1.4 Porovnání knihoven

Výše uvedené knihovny pro tvorbu uživatelského rozhraní byly vybrány na základě své podobnosti a jejich všeobecné známosti. Přímé porovnání knihoven, resp. frameworků není bez dalšího a hlubšího zkoumání možné provést. O výběru budoucí knihovny mohou rozhodnout některé níže uvedené ukazatele.

Angular a Vue mají vlastní CLI (Command Line Interface), přes který je možné jednoduše vytvořit základní aplikaci, tzv. „Hello World“. React tuto možnost nemá, ale vytvořil speciální generátor *create-react-app*, který základní aplikaci vygeneruje. Generátor se spustí pomocí speciálního příkazu *npx* softwaru package manageru NPM (Node Package Manager). Na následujícím grafu je vidět velikost vytvořených základních aplikací. Nutno podotknout, že Angular je komplexní framework a tomu také odpovídá jeho velikost.

Tabulka 2 Velikost základních aplikací



Zdroj.: vlastní zpracování

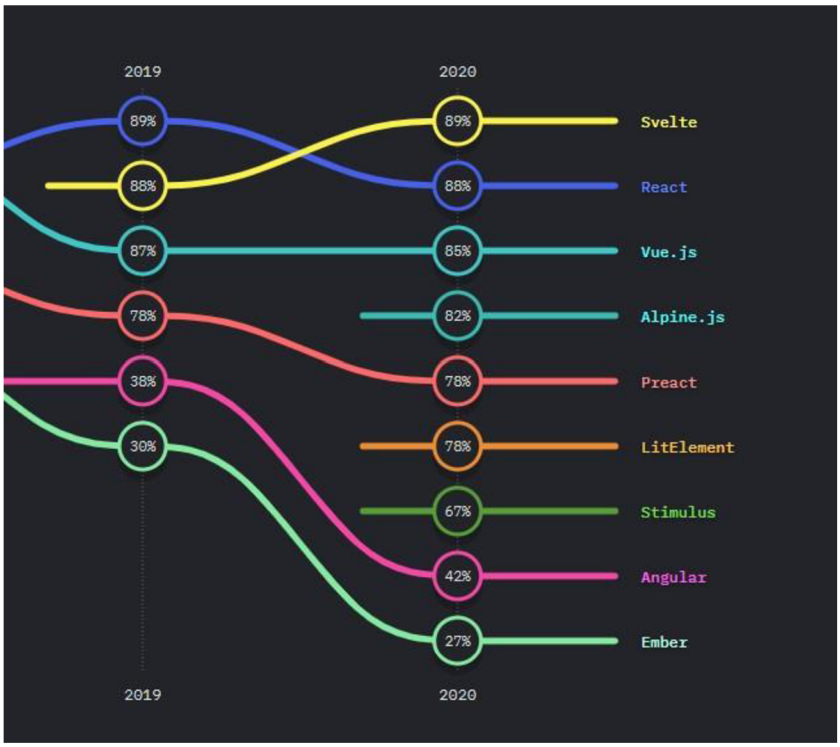
Pohledem na oblíbenost dle hodnocení na platformě Github je zjištěno, že knihovna Vue (16) s 169 tisíci hvězd vede oproti kihovně React (17) , která má 153 tisíc hvězd. Daleko za těmito knihovnami se nachází Angular (18) s 63,8 tisíci hvězdami. Hvězdy jsou vyjádřením obliby uživatelů daného projektu.

K obdobnému srovnání docházejí autoři State of JavaScript, kteří v rámci průzkumu ekosytému javascriptu v roce 2019 oslovili 21 717 respondentů (19). Jak vyplývá z průzkumu front-end kihoven uvedených na stránkách 2019.stateofjs.com (20), s knihovnou React je spokojeno 89 % uživatelů, s knihovnou Vue je spokojeno 87 % uživatelů a s frameworkem Angular je spokojeno 38 % uživatelů (20). Přičemž nejvíce uživatelů používá React (16099), následuje Angular s počtem 11 582 a posledním Vue s počtem 9320 uživatelů.

Obr. 2 Hodnocení zkušeností s front-end frameworkem v roce 2019

Zdroj: State of JS (20)

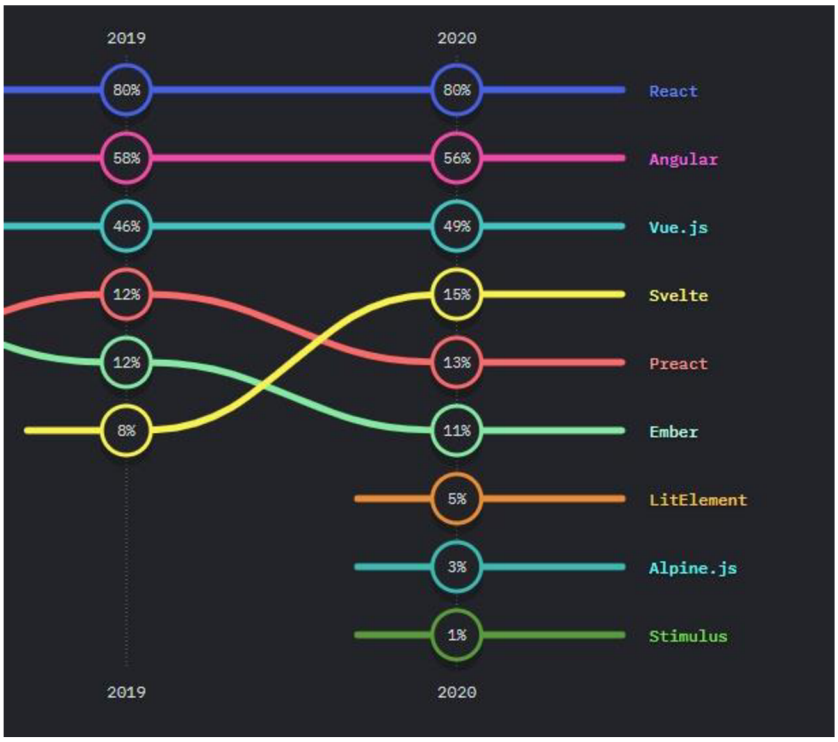
V roce 2020 proběhl další veřejný průzkum provedený autory State of JS (21) , kterého se zúčastnilo 23 765 lidí ze 137 zemí z celého světa. Srovná-li se výsledek průzkumu oblíbenosti front-end knihovny v roce 2020 (22) lze usuzovat, že se nekonají žádné převratné změny. Vedoucí pozici ovládla knihovna Svelte, která o 1 % převýšila knihovnu React. Obě tyto knihovny jsou následované knihovnou Vue. Oproti roku 2019 se v průzkumu objevilo několik nových knihoven, Alpine.js, LitElement a Stimulus.



Obr. 3 Hodnocení zkušeností s front-end frameworkem v roce 2020

Zdroj: State of JS (22)

Nicméně oblíbenost jednotlivých knihoven je pouze jedna část ukazatele. Dalším důležitým ukazatelem, který je obsažen v průzkumu front-end knihoven (22), je míra použití jednotlivých knihoven v roce 2020. Míra použití představuje poměr mezi těmi, kteří knihovnu použili a už by ji na dále nepoužívali, plus těmi, kteří by knihovnu použili opakovaně, a celkovým počtem odpovědí. Z výsledků je patrné, že knihovna React je velmi stabilní a znovu používaná knihovna pro vývoj SPA.



Obr. 4 Hodnocení použití front-end frameworku v roce 2020

Zdroj: State of JS (22)

Autor na začátku psaní této práce měl minimální znalost jazyka Javascript a žádné zkušenosti s výše uvedenými knihovnami. Může tedy subjektivně zhodnotit učící křivku s jednotlivými knihovnami. První zkušenost byla s knihovnou React. Vzhledem k malé znalosti jazyka Javascript byly některé syntaktické konstrukce těžší na pochopení. Po lepším porozumění jazyku Javascript autor cítil větší pochopení myšlenky knihovny, a během relativně krátké doby byl schopný implementovat složitější uživatelské rozhraní včetně použití knihoven třetích stran. S rostoucí znalostí jazyka Javascript se stávalo používání knihovny React snazší a intuitivnější, a to zejména přechodem na čistě funkční komponenty využití vlastnosti Reactu a jeho háčků.

Druhou knihovnou, se kterou měl možnost autor pracovat, byl Angular. Ta využívá jazyka Typescript, který je nástavbou jazyka Javascript, a bylo proto nutné si doplnit znalosti ohledně nové syntaxe. Angular je robustní řešení pro tvorbu aplikace a svojí komplexností převyšuje ostatní dvě srovnávané knihovny. Tomuto faktu také odpovídá pomalejší učící křivka, která odráží množství informací potřebných pro

základní pochopení fungování knihovny. Uživatel je do jisté míry nucen adoptovat filozofii knihovny.

Poslední knihovnou je Vue. Konceptuálně se jedná o mix přístupů obou výše uvedených knihoven. Vzhledem ke zkušenosti s ostatními knihovnami byla učící křivka znatelně rychlejší. Pochopení základních principů byla otázka krátké doby. Zapouzdření odpovědnosti komponenty do jednoho souboru zpřehledňuje vývoj.

Jak z textu vyplývá, subjektivní učící křivka je do velké míry ovlivněna externími faktory jako je například znalost programovacího jazyka nebo zkušenost s jinými podobnými knihovnami. Dalším faktorem je samotná komplexnost knihoven a jejich účel, resp. použití.

Autor si vybral pro implementaci klienta aplikace knihovnu React, a to zejména z důvodu podpory knihoven třetích stran, jako je například Bootstrap nebo Traverson. Dalším faktorem při výběru bylo velké množství aktivních uživatelů knihovny.

4.2 Přehled výběru dostupných technologií pro tvorbu serveru

V následující kapitole bude představeno několik frameworků pro tvorbu serveru v programovacím jazyce Java. Ostatní populární frameworky pro tvorbu serveru v jiných programovacích jazycích, například Nodejs a Express, zde nejsou uvedeny. Již na začátku psaní této části si autor stanovil, že serverová část bude naprogramována v jazyce Java.

4.2.1 Dropwizard

Dropwizard je open source Java framework, který umožňuje vývoj REST aplikace. Jak se uvádí na oficiálních stránkách frameworku (23) jedná se o kolekci pečlivě vybraných knihoven, které ulehčují vývoj REST aplikace nebo dodávají hotové řešení například pro konfiguraci nebo logování. Framework si klade za cíl doručit webovou službu v produkční kvalitě za co nejkratší možný čas.

Každá webová aplikace běží na nějakém druhu serveru. Dropwizard pro tento účel využívá knihovnu Jetty, která po spuštění aplikace vytvoří vložený server. Svojí funkcí odpovídá serveru vytvořenému Spring boot aplikací, která bude popsána dále v textu.

Uživatelský návod na stránkách frameworku Dropwizard (24) popisuje další důležité knihovny, které používá k dosažení cíle. Jersey knihovna, která implementuje JAX-RS referenci, se využívá pro vytvoření REST webové aplikace. Mapuje HTTP požadavky na jednoduché Java objekty.

Dále využívá například knihovny Jackson, který dokáže převést Java objekt do formátu JSON a zpět. Tomuto procesu se říká serializace a deserializace nebo také mapování. Zkratka JSON podle standardu ECMA-404 (25) znamená JavaScript Object Notation. *„JSON je lehká, textově založená, jazykově nezávislá syntaxe pro definování formátů výměny dat. Je odvozena z ECMAScript programovacího jazyka, ale je nezávislá na programovacím jazyce. JSON definuje malou strukturální množinu pravidel pro přenosnou reprezentaci strukturovaných dat.*“[10] ECMA 404 (25)(překlad autora). V dnešní době je JSON prakticky standardem pro výměnu dat po síti.

Mezi další zajímavé knihovny, které využívá framework Dropwizard, patří Logback a slf4j pro logování. JBDI pro komunikaci s relační databází. Mustache šablonovací systém.

Autor s touto knihovnou nepracoval a není proto schopen zhodnotit její využití.

[10] JSON is a lightweight, text-based, language-independent syntax for defining data interchange formats. It wasderived from the ECMAScript programming language, but is programming language independent. JSONdefines a small set of structuring rules for the portable representation of structured data.

4.2.2 Micronaut

Micronaut framework se na oficiálních stránkách Github (26) popisuje jako moderní, plnohodnotný nástroj, založený na JVM (Java Virtual Machine) pro tvorbu lehce testovatelné aplikace. Nabízí podporu hned několika jazyků Java, Kotlin a Groovy.

Javu není nikterak potřeba představovat. Programovací jazyk Kotlin byl vyvinut firmou JetBrains jak je uvedeno na oficiálních stránkách jazyka (27). Jedná se o open source staticky typový programovací jazyk, který je schopný fungovat na platformách JVM, Android, a Native. Lze jej využít pro objektově-orientované programování, ale i pro funkcionální programování. Kotlin je interoperabilní s jazykem Java a lze jej plynule integrovat do stávajícího kódu nebo v jazyce Kotlin používat konstrukce z jazyku Java.

Oficiální stránky programovacího jazyka Groovy (28) popisují programovací jazyk Groovy jako staticky typový, určený pro platformu JVM. Vyvinut byl společností Apache s cílem zlepšit produktivitu vývojáře tím, že mu poskytnou povědomou a lehce zapamatovatelnou syntaxi. Lze jej integrovat do jakéhokoliv Java programu.

Micronaut framework je inspirován jinými frameworky jako jsou Spring, Spring Boot a Grails. Poskytuje nástroje například DI (Dependency Injection), IoC (Inversion of Control) nebo auto-konfiguraci. Dále obsahuje nástroje důležité při vývoji mikro služeb jako jsou HTTP Routing , Service Discover nebo Client-Side Load Balancing.

I když je znatelná inspirace ostatními frameworky, Micronaut se snaží eliminovat nevýhody ostatních frameworků například minimálním použitím reflexe.

Autor s tímto frameworkem nepracoval a není proto schopen dostatečně zhodnotit jeho využití.

4.2.3 Spring framework

Spring framework je velmi rozšířený a velmi populární framework určený pro platformu Java. Pracuje na principu Inversion of Control a v jeho implementaci IoC kontejneru, který vytvoří kontext aplikace a řídí závislosti v rámci kontejneru (29). Do kontextu aplikace jsou vytvořeny instance tříd, které jsou potřeba v jiných třídách. Lépe řečeno, samotná třída, která potřebuje využít služby jiné třídy, si nevytváří novou instanci sama. Pouze deklaruje, jakým způsobem jí má být potřebná třída dodána. Jedná se o takzvané vsazování závislosti (dependency injection). Spring framework k tomuto účelu používá tři způsoby. Vsazení závislosti pomocí konstruktoru třídy (Constructor injection). Tento způsob je autory Springu nejvíce doporučován. Dále pak existuje vsazení závislosti přes setter metodu (Setter injection). Posledním způsobem je vsazení závislosti přes deklarovaný atribut v třídě (Field injection). Tento poslední způsob naopak není autory Springu doporučován.

Na příkladu (**Chyba! Nenalezen zdroj odkazů.**) je ukázáno použití vsazení závislosti přes konstruktor třídy. Třída SumWorkAttendanceController deklaruje použití dvou soukromých atributů. Jedná se o reference na pomocné třídy poskytující určitý servis. Jak je patrné z ukázky kódu, nikde nedochází k vytvoření konkrétní instance požadovaných tříd. Vytvoření konkrétní instance třídy, například WorkTimeSumCounter, je odpovědností IoC kontejneru. Konstruktor třídy není však v ukázce kódu nikde zmíněn. Toto není chybou, ale důsledek využití notace @AllArgsConstructor externí knihovny nesoucí název Project Lombok, zkráceně Lombok.

```
@BasePathAwareController
@AllArgsConstructor
public class SumWorkAttendanceController {

    private final WorkTimeSumCounter workTimeSumCounter;
    private final PremiumPaymentsSumCounter premiumPaymentsSumCounter;

    @GetMapping(path = "work-attendances/{id}/work-time-sum")
    public @ResponseBody
    ResponseEntity<?> getWorkTimeSum(@PathVariable("id") WorkAttendance
workAttendance) {
        final WorkTimeSum workTimeSum =
workTimeSumCounter.calculate(workAttendance);
        return ResponseEntity.ok(workTimeSum);
    }
    // vynecháno

}
```

Ukázka kódu 1 Ukázka vsazování závislosti přes konstruktor s využitím Project Lombok

Zdroj: vlastní zpracování

Lombok je knihovna, která eliminuje tzv. boilerplate code, tedy kód, který se často opakuje beze změny jeho variace, přičemž nejde o duplikaci kódu. I přesto, že v ukázce **Chyba! Nenalezen zdroj odkazů.** chybí konstruktor, jedná se pro kompilátor o validní zápis. Před samotnou kompilaci třídy knihovna Lombok doplní nezbytný kód do třídy. Výsledkem je tak lépe čitelný zápis třídy. Nejvíce je tento efekt vidět na ukázce **Chyba! Nenalezen zdroj odkazů.** kde je navržena datová třída s názvem WorkTime. V třídě jsou pouze deklarované atributy. Veškeré další věci zajišťuje knihovna Lombok pomocí notace @Data (30). Tedy vygenerování settrů a gettrů pro všechny atributy, implementace metod toString, equals a hasCode a v neposlední řadě konstruktor třídy.

```java
@Data
@Entity
public class WorkTime {

    @Id
    @GeneratedValue
    private long workingTimeId;

    /**
     * Celková délka směny
     */
    private double length;
    /**
     * Odpracované hodiny ve smyslu fyzické přítomnosti na pracovišti.
     * Započitatelné do pracovní doby
     */
    private double workedOut;
    /**
     * Neodpracované hodiny ve smyslu hodin započitatelných do pracovní
     * doby, které nebyly fyzicky odpracované (např. neschopnost)
     */
    private double notWorkedOut;

    /**
     * Hodiny, které se považují za dovolenou
     */
    private double holiday;
}
```

Ukázka kódu 2 Ukázka datové/entitní třídy s použitím Lombok

Zdroj: vlastní zpracování

Spring Framework byl vyvinut v roce 2003 (31), od té doby doznal značných změn. Zejména samotný framework slouží jako platforma pro další projekty. Asi mezi nejzásadnější projekty lze zařadit Spring Boot, který abstrahuje složité nastavování vnitřního stavu frameworku. Spring Boot postupně přebral roli platformy a zpravidla pokud se někde mluví o Spring, je tím myšlen právě projekt Spring Boot.

Jak bylo řečeno, Spring je rozdělen do několika separátně se vyvíjejících modulů neboli projektů. Tyto projekty dovolují vývojářům rozšiřovat funkcionality svého projektu během vývoje. Mezi známější projekty lze zařadit Spring Security, Spring Data, Spring Cloud, Spring Web a Spring HATEOAS.

Pro implementaci serverové části aplikace byl vybrán framework Spring. Jak bylo naznačeno v představení frameworku, jedná se o modulární framework, který

umožňuje přidávat funkcionality v průběhu vývoje aplikace. Tyto moduly přináší řešení tzv. out of the box dané funkcionality s minimální konfigurací.

5 Vybrané aspekty implementace

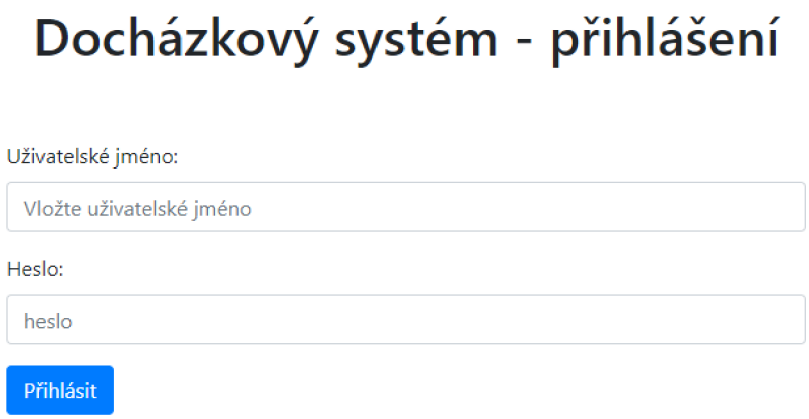
Aplikaci tvoří dva samostatné celky. Frontend – klientská část aplikace, a backend – serverová část aplikace. Každá z těchto částí je tvořena jinými technologiemi, které budou popsány u jednotlivých částí. Aplikace byla navržena a naprogramována s ohledem na co možná nejnižší provázání frontendu a backendu.

5.1 Frontend

Klientská část aplikace, frontend, je vytvořena jako SPA aplikace postavená na technologii React ve verzi 17.0.1. Z pohledu úplnosti návrhu a implementace uživatelského rozhraní se jedná o funkční prototyp. Dalšími významnějšími technologiemi, které klientská část využívá, jsou React Query ve verzi 3.5.11, React Bootstrap ve verzi 1.4.0, Traverson-hal ve verzi 7.0.2 a Traverson-promise ve verzi 0.0.9.

5.1.1 Přihlášení uživatele

Dle typové úlohy uvedené v 3.3.1 je vytvořena přihlašovací stránka, která zároveň slouží jako úvodní obrazovka aplikace. Po přihlášení je uživatel přesměrován na editační okno.



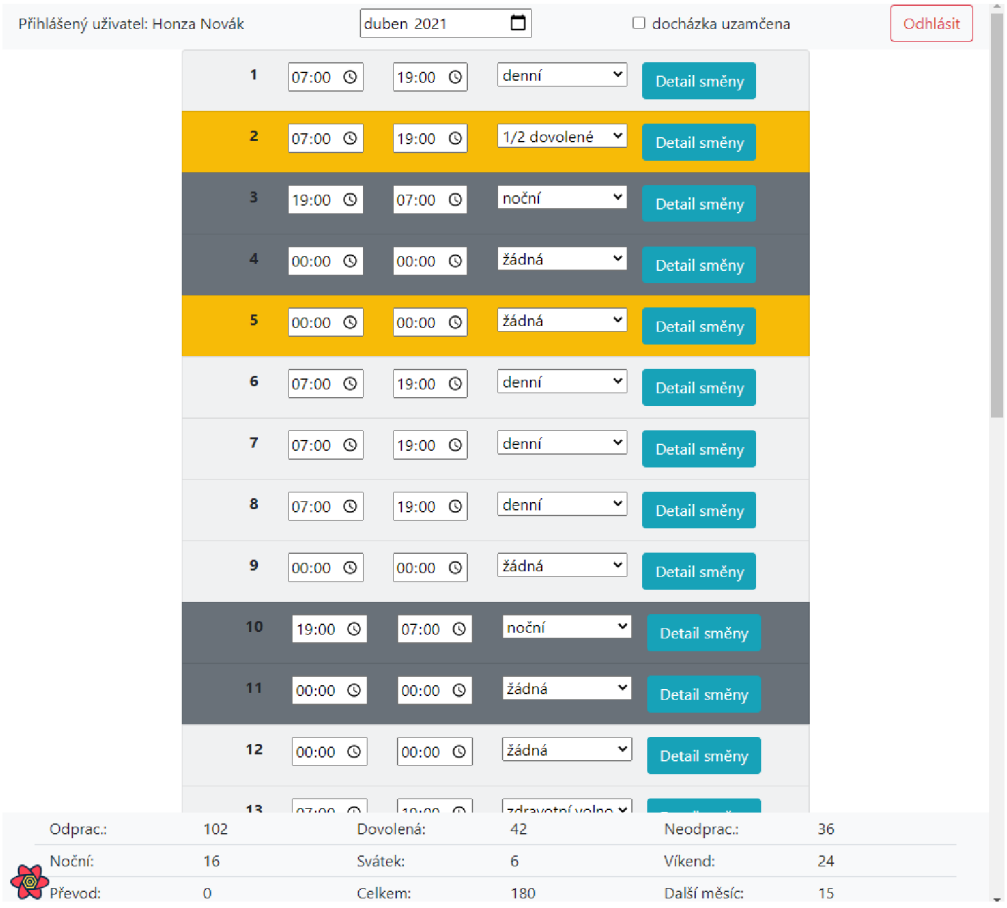
Obr. 5 Úvodní obrazovka uživatelského rozhraní – přihlášení

Zdroj: vlastní zpracování

V prototypu není zabezpečení implementováno, proto je možné se přihlásit vyplněním jakékoliv hodnoty. Obě pole však musí být vyplněna. Stav přihlášení uživatele je provizorně realizován zápisem do session storage jako klíč s názvem loggingStatusKey s boolean hodnotou. Možné zabezpečení aplikace je zmíněno v závěru 6.

5.1.2 Editační okno

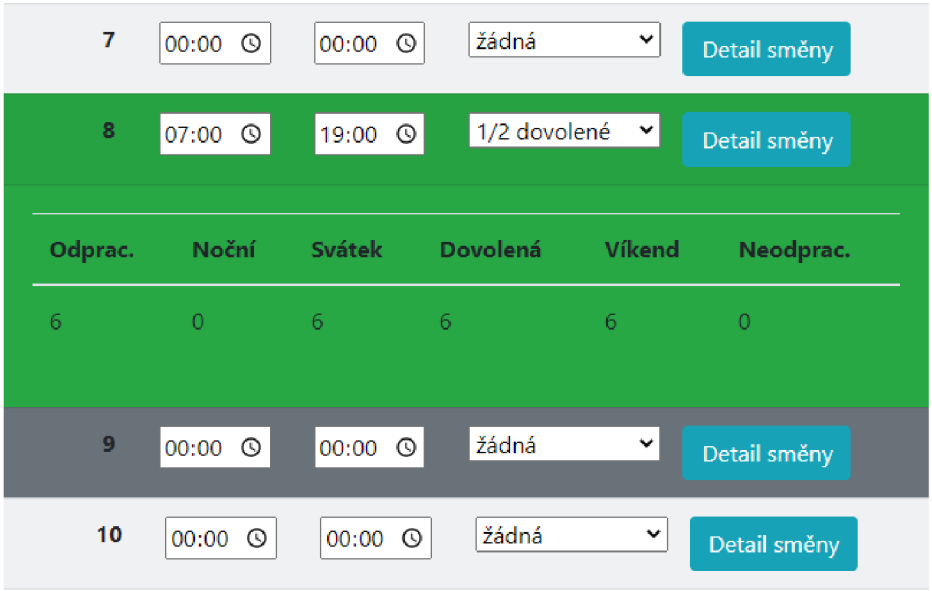
Dle typové úlohy uvedené v 3.3.2 a 3.3.3 je vytvořeno editační okno, které umožňuje zobrazit pro jednotlivé měsíce pracovní docházku. U horního okraje okna je zobrazen navigační panel, který obsahuje informace o přihlášeném uživateli, ovládací prvek pro výběr datumu, ovládací prvek pro uzamčení a odemčení docházky a ovládací prvek pro odhlášení.



Obr. 6 Editační okno uživatelského rozhraní

Zdroj: vlastní zpracování

Pro každý den v měsíci je v editačním okně vytvořen řádek s možností editace. Za číselným označením dne se nacházejí dva ovládací prvky pro editaci začátku a konce směny. Při ovládání vstupu začátku nebo konce směny šipkami klávesnice se číselníky posouvají u hodnoty hodin o jedna a u hodnoty minut o 15. Dále je zde možnost výběru z definovaných typů směn. Dojde-li ke změně typu směny a provede se komunikace se serverovou částí a začátek a konec směny je aktualizován na základě zvoleného typu. Jakým způsobem probíhá komunikace se serverovou částí bude zmíněno dále v textu. Omezení v podobě uzamčené pracovní docházky jsou přítomna v obou částech aplikace. Posledním ovládacím prvkem je tlačítko, které zobrazuje detail směny pro daný den.



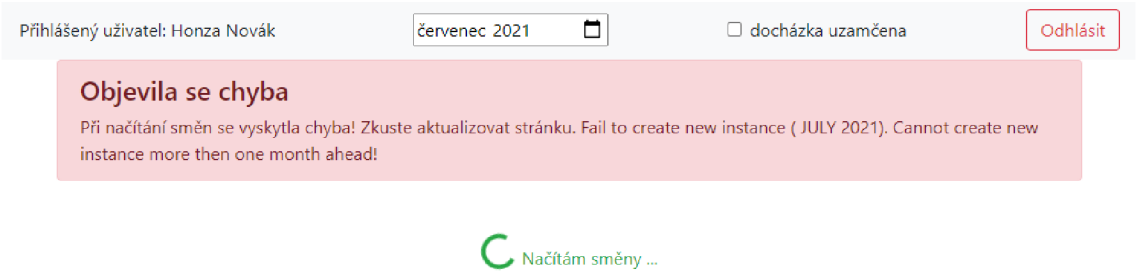
Obr. 7 Editační okno uživatelského rozhraní – detail směny

Zdroj: vlastní zpracování

Dle požadované specifikace jsou dny rozlišeny podle druhu. Světle šedá barva značí všední den, tmavě šedá víkend. Žlutou barvou je označen svátek připadající na všední den. Zelenou barvou je označen svátek připadající na víkendový den.

Omezení zobrazení docházky definované ve specifikaci 3.3.2, je implementováno na straně serveru, a v případě nemožnosti vytvoření docházky je v uživatelském rozhraní zobrazena chybová zpráva.

Založení nové docházky je provedeno pomocí ovládacího prvku pro výběr datumu. Splňuje-li vybrané datum definovaná omezení, zobrazí se nově vytvořená pracovní docházka. V opačném případě je v uživatelském rozhraní zobrazena chybová zpráva. Požadavek na autorizovaného uživatele není implementován.



Obr. 8 Editační okno uživatelského rozhraní – chybová zpráva

Zdroj: vlastní zpracování

Komunikace se serverovou částí aplikace probíhá prostřednictvím knihovny React Query (32), která je využita pro synchronizaci dat se serverem (fetching and updating data), ukládání do mezipaměti (caching) a řízení globálních proměnných (global state management). Hlavním nástrojem je hook useQuery, který přijímá jako jeden z parametrů funkci vracející promise, pomocí níž asynchronně načte data a uloží je do mezipaměti pod definovaný klíč. Data jsou pak přístupná právě pomocí klíče (query key). Query key je unikátní identifikátor dat. Data se během svého životního cyklu mohou nacházet ve čtyřech stádiích.

$$fetching \rightarrow fresh \rightarrow stale \rightarrow inactive$$

Obr. 9 Diagram přechodu stavu dat v React Query

Zdroj: vlastní zpracování

Fetching v případě asynchronního načítání data. Fresh, pokud jsou data aktuální. Stale stádium označuje data, která jsou považována za zastaralá. Posledním stádiem je inactive. Jsou-li data neaktivní po předem definovanou dobu, jsou vyřazeny z mezipaměti.

Dalším velmi důležitým komunikačním nástrojem mezi klientem a serverem je knihovna Traverson (33), respektive její nástavba Traverson-promise. Tato knihovna zpracovává JSON ve formátu HAL.

„Norma JSON Hypertext Application Language (HAL) je standard, který zavádí konvence pro vyjádření hypertextových odkazů pomocí JSON [RFC4627].“[11] M. Kelly (34) (vlastní překlad). Definuje, jakým způsobem je strukturovaný JSON formát a jakým způsobem by se měl jeho obsah konzumovat.

```
GET /orders/523 HTTP/1.1
Host: example.org
Accept: application/hal+json

HTTP/1.1 200 OK
Content-Type: application/hal+json

{
  "_links": {
    "self": { "href": "/orders/523" },
    "warehouse": { "href": "/warehouse/56" },
    "invoice": { "href": "/invoices/873" }
  },
  "currency": "USD",
  "status": "shipped",
  "total": 10.20
}
```

Ukázka kódu 3 – Ukázka HAL formátu

Zdroj: (34)

Knihovna Traverson se používá pro *„konzumaci REST API, která se řídí zásadou HATEOAS, tj. rozhraní REST API, které mají vazby, odkazy, (links) mezi svými prostředky. Takové API, někdy také označované jako hypermedia nebo hypertext-driven API, má obvykle kořenový zdroj/koncový bod, který odkazuje na jiné*

[11] The JSON Hypertext Application Language (HAL) is a standard which establishes conventions for expressing hyperlinks with JSON [RFC4627].

zdroje/koncové body."[12] Traverson (35) (vlastní překlad). Pomocí knihovny Traverson lze načíst kořenový koncový bod, který obsahuje odkazy na další zdroje. Jak název knihovny naznačuje můžeme přecházet (travers) mezi jednotlivými odkazy k požadovanému zdroji, jak je vidět na následující ukázce.

```
function getAllUsers(url) {
    return traverson.from(url)
        .jsonHal()
        .follow("employees", "employees[$all]")
        .getResource().result
}
```

Ukázka kódu 4 - Použití knihovny Traverson-promise

Zdroj: vlastní zpracování

Metoda follow obsahuje parametry, podle kterých bude procházet po jednotlivých odkazech. Provolá se kořenový koncový bod, který vrátí odpověď jako JSON ve formátu HAL. Traverson nalezne objekt _links, ve kterém hledá shodu s předaným argumentem v metodě follow. Najde-li shodu, vyhledá odkaz pod klíčem href a pokračuje iterativním způsobem dále.

[12] ... consuming REST APIs that follow the HATEOAS principle, that is, REST APIs that have links between their resources. Such an API (also sometimes referred to as hypermedia or hypertext-driven API) typically has a root resource/endpoint, which publishes links to other resources.

```
// kořenový koncový body
{
  "_links": {
    "employees": {
      "href": "http://localhost:8080/api/v1/employees{?page,size,sort}",
      "templated": true
    },
       … // vynecháno
    "profile": {
      "href": "http://localhost:8080/api/v1/profile"
    }
  }
}
```

Ukázka kódu 5 - Odpověď po provolání kořenového koncového bodu

Zdroj: vlastní zpracování

V předchozím textu byl zmíněn pojem HATEOAS (Hypermedia As The Engine Of Application State) v souvislosti s REST API. Tento pojem je důležitý a do větší míry určoval implementaci zadaného problému. Roy Thomas Fielding (36) tento pojem zmiňuje ve své disertační práci popisující architekturu REST. Jedná se o jedno ze 4 omezení rozhraní. Více tento pojem upřesňuje na svém blogu (37), kde píše, že uživateli REST API by měla postačovat znalost kořenového koncového bodu. Další možnosti nakládání se zdrojem jsou odvíjeny od dostupnosti (a affordance) odkazů. Překlad slova a affordance, jako dostupnost není zcela přesný. Je tím myšlen význam, že pokud je pozorován objekt, je hned patrné, jak s objektem můžeme nakládat. Příkladem může být hrnek. Už pouhým pohledem je možné odhadnout jakým způsobem jej lze uchopit a co s ním lze dělat. V affordance lock je typickým příkladem principu HATEOAS. Lock slouží k přechodu z jednoho stavu do druhého prostřednictvím odkazu vztahující se k jedné konkrétní entitě.

```
// http://localhost:8080/api/v1/work-attendances/1907
{
  // vynecháno
  "_links": {
    // vynecháno
    "work-time-sum": {
      "href": "http://localhost:8080/api/v1/work-attendances/1907/work-time-sum"
    },
    "premium-payments-sum": {
      "href": "http://localhost:8080/api/v1/work-attendances/1907/premium-payments-sum"
    },
    "lock": {
      "href": "http://localhost:8080/api/v1/work-attendances/1907/lock"
    },
    // vynecháno
  }
}
```

Ukázka kódu 6 Affordance pro zdroj work-attendances v HAL formatu

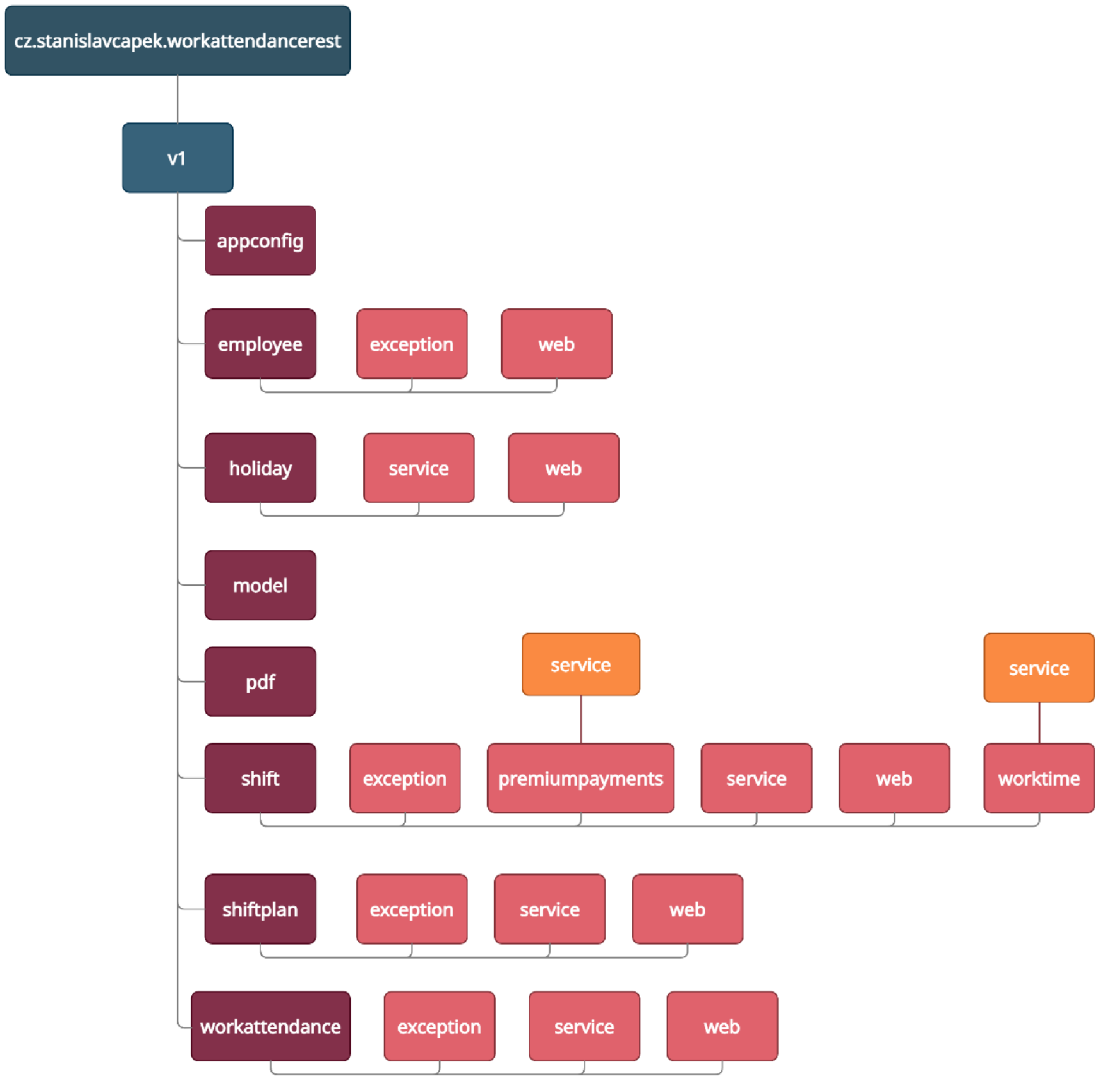
Zdroj: vlastní zpracování

V této kapitole bylo popsáno uživatelské prostředí, jeho vzhled a funkčnost. Velká část byla věnována vysvětlení pojmů související s konzumací REST API a práce s formátem HAL. Byly vysvětleny podpůrné technologie React Query a Traverson.

5.2 Backend

Serverová část aplikace, backend, je spravována nástrojem pro automatizaci sestavení (a build automation tool) Maven (38), který poskytuje uniformní strukturu projektu. Automaticky spravuje definované závislosti využívané v projektu. Existuje Maven repositář projektů, ze kterých lze vybírat a volně jej používat. Jeden z těchto projektů je Spring Boot ve verzi 2.4.0, na kterém je postavena serverová část aplikace. Spring Boot, jak bylo popsáno v 4.2.3, tvoří pouze základ aplikace. Aplikace využívá další projekty z rodiny Spring. Pro vytvoření serverové aplikace je použit projekt Spring Web, který vytváří architekturu MVC s vloženým Apache Tomcat serverem. Pojem MVC je vysvětlen v 4.1.2. Spring Data REST pro správu entit a CRUD operací nad těmito entitami. Tomuto projektu, dále v textu, bude věnována větší pozornost. Pro účely vývoje je použita relační SQL databáze H2.

5.2.1 Struktura



Obr. 10 Struktura balíčků projektu

Zdroj: vlastní zpracování

Diagram struktury (Obr. 10) popisuje rozdělení aplikace do jednotlivých balíčků podle entit. Každý balíček agreguje třídy vztahující se k dané entitě a dále se dělí dle funkčního zařazení. Například entita Holiday obsahuje balíček service a web. Balíček service obsahuje třídu přímo pracující s entitou nebo poskytující entitě službu. V tomto případě načtení svátků z externího zdroje. Balíček web obsahuje třídy, které se vztahují k vystavení entity vnějšku. Mohou sem patřit například kontroléry nebo repositář.

5.2.2 Fungování Spring Data REST a Spring HATEOAS

Referenční příručka Spring Data REST (39) uvádí přesný a výstižný popis daného projektu: *„Spring Data REST je založeno na Spring Data repositářích a automaticky je exportuje jako zdroje REST. Využívá hypermedii k tomu, aby klienti mohli automaticky vyhledávat funkce vystavené repositářem a integrovat tyto zdroje do souvisejících funkcí založených na hypermediích.“*[13](překlad autora). Autoři projektu Spring Data REST se inspirovali Domain-Driven Design přístupem, který pospal Eric Evans ve stejnojmenné knize. Entita/zdroj je objekt, který je určitelný dle jeho identity, která je nezměnitelná po celou dobu jejího života (40). V objektově orientovaném programování tyto entity odrážejí skutečné objekty s jistou nezbytnou mírou abstrakce. Bližší vysvětlení, co je entitou z pohledu Domain-Driven Design je nad rámec této práce.

Z definice Spring Data REST vyplývá, že knihovna sama vystavuje koncový bod pro entitu, která má veřejný repositář.

```
public interface HolidayRepository extends JpaRepository<Holiday, Long> {

    Holiday findByDate(LocalDate date);

    boolean existsByDate(LocalDate date);

    @Query(value = "select case when exists (select * from holiday where
year(date) = ?1) then true else false end",
            nativeQuery = true)
    boolean existsByDateYear(int year);
}
```

Ukázka kódu 7 Definování HolidayRepository v Spring Data REST

Zdroj: vlastní zpracování

[13] Spring Data REST builds on top of the Spring Data repositories and automatically exports those as REST resources. It leverages hypermedia to let clients automatically find functionality exposed by the repositories and integrate these resources into related hypermedia-based functionality.

Pro vystavení zdroje postačuje rozšířit interface JpaRepository<T, ID>, resp. CrudRepository<T, ID>. Jakákoliv další implementace není nutná. Spring automaticky implementaci poskytne. V Ukázka kódu 7 Definování HolidayRepository v Spring Data REST jsou definované metody findByDate a existsByDate, které sledují jmennou konvenci, a proto je Spring schopen analyzovat a vytvořit odpovídající implementaci. Pokud by se z nějakého důvodu nepodařilo takovou metodu sestavit, existuje anotace @Query, která dovoluje vložit vlastní SQL dotaz.

Odstranění potřeby vytvářet implementaci pro jednoduché CRUD operace nad databází je pouze část výhody využití Spring Data REST. Pro takto definovaný interface HolidayRepository dodá Spring také kontrolér s příslušnými koncovými body.

Ve frameworku Spring obecně platí, že téže věc lze dělat několika způsoby. Následující ukázka je jedna z možností, jakým způsobem je možné omezit http metody nad jednotlivými zdroji. Pro zdroj Holiday, se v ukázce omezují POST, PATCH, PUT a DELETE.

```
@Autowired
public void configure(
        RepositoryRestConfiguration repositoryRestConfiguration) {

    final ExposureConfiguration config = repositoryRestConfiguration.getEx-
posureConfiguration();

    config.forDomainType(WorkAttendance.class).disablePutForCreation();
    config.forDomainType(Shift.class).disablePutForCreation();
    config.forDomainType(Employee.class).disablePutForCreation();

    ExposureConfigurer.AggregateResourceHttpMethodsFilter disableModifica-
tionMethods = (metadata, httMethods) -> {
        return httMethods.disable(HttpMethod.POST, HttpMethod.PATCH, Http-
Method.PUT, HttpMethod.DELETE);
    };

    config.forDomainType(Holiday.class)
            .withItemExposure(disableModificationMethods)
            .withCollectionExposure(disableModificationMethods);
}
```

Ukázka kódu 8 Omezení http metod v Spring Data REST

Zdroj: vlastní zpracování

Implicitní kontrolér lze přepsat a definovat si namísto něho vlastní. K tomuto účelu slouží Spring Data REST anotace @RepositoryRestController (39). Takto anotovaný kontrolér získává výhody správy výjimek, konverze zpráv, znalost základní cesty ke koncovým bodům a možnosti volně implementovat CRUD operace.

Další anotace Spring Data REST @BasePathAwareController (39) se využívá v případech, kdy postačuje implicitní kontrolér pro CRUD operace a je potřeba zdroj rozšířit o novou funkcionalitu. Kontrolér získává znalost základní cesty ke koncovým bodům. V Ukázka kódu 9 je znázorněna implementace přechodu stavu zamčeno pro pracovní docházku.

```
@BasePathAwareController
@AllArgsConstructor
public class LockWorkAttendanceController {

    private final WorkAttendanceRepository repository;

    @PostMapping(path = "work-attendances/{id}/lock")
    public @ResponseBody
    ResponseEntity<?> lockWorkAttendance(@PathVariable("id") WorkAttendance
workAttendance,
                                         @RequestBody LockStatus status,
                                         PersistentEntityResourceAssembler
assembler) {

        workAttendance.setLocked(status.isLocked());
        final WorkAttendance saved = repository.save(workAttendance);
        final PersistentEntityResource model =
assembler.toFullResource(saved);
        return ResponseEntity.ok(model);
    }

    @Value
    @AllArgsConstructor(onConstructor = @__(@JsonCreator))
    private static class LockStatus {
        boolean locked;
    }
}
```

Ukázka kódu 9 BasePathAwareController pro WorkAttendance entitu

Zdroj: vlastní zpracování

Aby bylo možné s entitami pracovat v případě použití implicitního kontroléru, Spring Data REST (39) vyvolává 8 událostí (BeforeCreateEvent, AfterCreateEvent, BeforeSaveEvent, AfterSaveEvent, BeforeLinkSaveEvent, AfterLinkSaveEvent,

BeforeDeleteEvent, AfterDeleteEvent). Implementováním obsluhy/posluchače pro tyto události lze docílit kontroly a případné aplikovaní validací či business logiky. V Ukázka kódu 10 se nachází implementace vlastní obsluhy. Třída je anotována @RepositoryEventHandler a na základě typu parametru u metody anotované příslušnou událostí Spring Data REST zařadí obsluhu k dané entitě.

```
@RepositoryEventHandler
@AllArgsConstructor
@Component
public class WorkAttendanceEventHandler {

    private final ShiftFactory factory;
    private final WorkAttendanceRepository repository;
    private final WorkAttendanceUtilService service;

    @HandleBeforeCreate
    public void handleCreation(WorkAttendance workAttendance) {
        checkOneMonthAheadCondition(workAttendance);
        checkOneMonthBackCondition(workAttendance);
        checkDuplicationCondition(workAttendance);
        checkValidState(workAttendance);
        prepare(workAttendance);
    }
    // vynecháno
}
```

Ukázka kódu 10 EventHandler pro entitu WorkAttendance v Spring Data REST

Zdroj: vlastní zpracování

Poslední důležitou zmínkou je využití principu HATEOAS. Obecný koncept HATEOAS je vysvětlen v 5.1.2. Spring Data REST interně využívá projekt Spring HATEOAS (41). Hlavní myšlenkou projektu je obohacení zdroje o hypermedia. Dosahuje toho pomocí odkazů, které odkazují přímo na entitu, kolekci či na jiné zdroje. Pomocí odkazů lze vystavit pro zdroj dodatečnou funkcionalitu, jak je nastíněno v Ukázka kódu 9. Odkazy lze za běhu aplikace ke zdroji přidávat či ubírat, a teoreticky lze aplikaci ovládat pouze pomocí odkazů.

Spring HATEOAS obaluje entitu do třídy EntityModel, která representuje výsledný zdroj v HAL formátu. Pro odchycení obalené entity před vrácením volajícímu lze implementovat RepresentationModelProcessor. Poté je možné na základě business logiky přidávat, ubírat nebo jinak modifikovat odkazy. V Ukázka kódu 11 se provádí

dodatečná validace hodin v pracovní docházce. Vyvstane-li nový požadavek na rozšiřující funkcionalitu nabízí se tento postup. Vytvoření dedikovaného kontroléru anotovaného @BasePathAwareController a přidání odkazu do EntityModelu.

```
@Service
@AllArgsConstructor
public class WorkAttendanceProcessor implements
RepresentationModelProcessor<EntityModel<WorkAttendance>> {

    private EntityLinks entityLinks;
    private WorkAttendanceUtilService service;

    @Override
    public EntityModel<WorkAttendance> process(EntityModel<WorkAttendance>
model) {

        final WorkAttendance workAttendance = model.getContent();

        service.setValidHoursInMonth(workAttendance);

        addLinks(model);

        return model;
    }
    // vynecháno
}
```

Ukázka kódu 11 ModelProcessor pro zdroj WorkAttendance v Spring HATEOAS

Zdroj: vlastní zpracování

6 Výsledky a závěr

Cílem práce bylo vytvoření aplikace Docházkový systém s využitím aktuálních nástrojů a popsat základní principy použitých technologií.

Práce byla rozdělena do tří stěžejních částí. Kapitola Analýza a návrh aplikace se zabývá legislativním rámcem problematiky docházkových systémů. Zejména jsou zde uvedeny vybrané části zákona upravující vztah zaměstnance a zaměstnavatele, pracovní dobu, rozvržení pracovní doby a jiné. Druhou kapitolou je Výběr a popis technologií. V té byla provedena analýza dostupných technologií pro tvorbu klienta a serveru. Pro obě části aplikace byli vybráni tři kandidáti na základě oblíbenosti, obecné známosti a celkové podpoře knihoven jako takových. Pro klientskou část aplikace byla vybrána knihovna React a pro serverovou část byl vybrán framework Spring. Poslední kapitola nazvaná Vybrané aspekty implementace shrnuje důležité koncepty využité v rámci klientské a serverové části.

Klientská část aplikace byla implementována jako SPA aplikace. Serverová část aplikace jako REST API. Typové úlohy byly specifikovány tak, že pokrývají obě části aplikace. Rozdělení bylo zvoleno na funkční a non-funkční požadavky. V rámci implementace jednotlivých typových úloh nedošlo u některých k úplnému naplnění požadavků. Typová úloha 3.3.1 Přihlášení uživatele byla vytvořena bez zabezpečení. Vstupní pole pro přihlášení musí být vyplněna. U typové úlohy 3.3.2 Zobrazení pracovní docházky nebyl dodržen požadavek pro autorizaci uživatele. Autorizace uživatele je úzce spojena s autentizací uživatele, která také nebyla implementována. Typová úloha 3.3.3 Úprava pracovní docházky byla splněna vyjma převodu hodin nad rámec fondu při vyrovnávacím období. Typová úloha 3.3.4 Archivace pracovní docházky byla splněna zcela. Typová úloha 3.3.5 Generování šablony plánu směn byla splněna zcela. Typová úloha 3.3.6 Načtení vyplněné šablony plánu směn byla pozměněna, kdy ze šablony plánu směn je načten pouze aktuální měsíc.

Zabezpečení aplikace pomocí autentizace a autorizace bylo zvoleno jako budoucí rozšíření. Toho by se mohlo dosáhnout pomocí projektu Spring Security (42), který umožňuje do již existujícího Spring projektu vložit zabezpečení s minimální úsilím.

V základu podporuje OAuth 2.0, JWT (JSON Web Token) a Opaque Token. Uvažovat by se mělo nad zabezpečením komunikace pomocí HTTPS s certifikátem vystaveným důvěryhodnou autoritou.

V případě nasazení jedné instance aplikace na server a užíváním větším počtem uživatelů, by mohlo způsobovat nežádoucí latenci či konkurenční přístupy do databáze. V takovém případě by bylo vhodné aplikaci upravit jako microservice s využitím Spring Cloud. Spring Cloud a přidružené projekty (43) tvoří nástroj pro tvoření aplikací podle návrhu microservice. Využívá k tomu například mechanismů vyvážení zátěže, service-to-service volání, směrování a registrace a zjišťování služby.

7 Reference

1. **Aktion - Moderní docházkový systém.** ***aktion.cz.*** **[Online] EFG CZ spol. s.r.o., 2021. [Citace: 20. 3 2021.] https://www.aktion.cz/produkty/dochazkovy-system.html.**

2. Docházkový systém - online docházka jednoduše - Alveno.cz. ***alveno.cz.*** **[Online] protadesign.cz, 2021. [Citace: 20. 3 2021.] https://www.alveno.cz/dochazkovy-system.**

3. Docházkové systémy | IMA s.r.o. ***ima.cz.*** **[Online] NET Production, s.r.o., 2021. [Citace: 20. 3 2021.] https://www.ima.cz/produkty/dochazkove-systemy/.**

4. DOCHÁZKA - RON Software | ron.cz. ***ron.cz.*** **[Online] RON Software, spol. s.r.o., 2021. [Citace: 20. 3 2021.] https://www.ron.cz/cz/dochazka/.**

5. Docházkové systémy. ***z-ware.cz.*** **[Online] Z-WARE, 2020. [Citace: 20. 3 2021.] https://www.z-ware.cz/dochazkove-systemy.**

6. Nonfunctional Requirements - Scaled Agile Framework. ***scaledagileframework.com.*** **[Online] Scaled Agile, Inc., 2021. [Citace: 14. 3 2021.] https://www.scaledagileframework.com/nonfunctional-requirements/.**

7. Occhino, Tom a Walke, Jordan. JS Apps at Facebook. ***www.youtube.com.*** **[Online] 5. 8 2013. [Citace: 13. 7 2020.] https://www.youtube.com/watch?v=GW0rj4sNH2w.**

8. facebook/react : A declarative, efficient, and flexible JavaScript library for building user interfaces. ***Github, Inc.*** **[Online] 8. 7 2020. [Citace: 27. 12 2020.] https://github.com/facebook/react.**

9. Facebook Inc. Components and Props – React. *www.reactjs.org.* [Online] 2020. [Citace: 13. 7 2020.] https://reactjs.org/docs/components-and-props.html.

10. —. Introducing Hooks – React. *www.reactjs.org.* [Online] 2020. [Citace: 13. 7 2020.] https://reactjs.org/docs/hooks-intro.html.

11. Flanagan, David. *JavaScript : the definitive guide.* [online] Sebastopol, Calif : O'Reilly Media, Inc., 2006. 0596554478, 9780596554477.

12. Traversy media. Angular Crash Course - YouTube. *www.youtube.com.* [Online] 23. 1 2019. [Citace: 14. 7 2020.] https://www.youtube.com/watch?v=Fdf5aTYRW0E.

13. Angular - Introduction to Angular concepts. *angular.io.* [Online] Google, 2020. [Citace: 27. 12 2020.] https://angular.io/guide/architecture.

14. Introduction — Vue.js. *www.vuejs.org.* [Online] [2020]. [Citace: 18. 7 2020.] https://vuejs.org/v2/guide/index.html.

15. Single File Components | Vue.js. *v3.vuejs.org.* [Online] [2020]. [Citace: 27. 12 2020.] https://v3.vuejs.org/guide/single-file-component.html#what-about-separation-of-concerns.

16. vuejs/vue: 🖖 Vue.js is a progressive, incrementally-adoptable JavaScript framework for building UI on the web.. *Github, Inc.* [Online] 2020. [Citace: 2. 8 2020.] https://github.com/vuejs/vue.

17. facebook/react : A declarative, efficient, and flexible JavaScript library for building user interfaces. *Github, Inc.* [Online] 8. 7 2020. [Citace: 13. 7 2020.] https://github.com/facebook/react.

18. angular/angular: One framework. Mobile & desktop. *Github, Inc.* [Online] 2020. [Citace: 2. 8 2020.] https://github.com/angular/angular.

19. The State of JavaScript 2019: Demographics. *State of JavaScript.* [Online] 2020. [Citace: 3. 8 2020.] https://2019.stateofjs.com/demographics/.

20. The State of JavaScript 2019: Front End Frameworks. *State of JavaScript.* [Online] 2020. [Citace: 4. 8 2020.] https://2019.stateofjs.com/front-end-frameworks/.

21. State of JS 2020: Demographics. *State of JavaScript.* [Online] 2021. [Citace: 13. 3 2021.] https://2020.stateofjs.com/en-US/demographics/.

22. State of JS 2020: Front-end Frameworks. *State of JavaScript.* [Online] 2021. [Citace: 13. 3 2021.] https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/.

23. Dropwizard Team. Home — Dropwizard. *dropwizard.io.* [Online] 2020. [Citace: 28. 12 2020.] https://www.dropwizard.io/en/latest/index.html.

24. —. Getting Started — Dropwizard. *dropwizard.io.* [Online] 2020. [Citace: 28. 12 2020.] https://www.dropwizard.io/en/latest/getting-started.html.

25. Ecma International. ECMA-404. *ecma-international.org.* [Online] 2017. [Citace: 28. 12 2020.] http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

26. micronaut-projects/micronaut-core: Micronaut Application Framework. *github.com.* [Online] [2020]. [Citace: 28. 12 2020.] https://github.com/micronaut-projects/micronaut-core.

27. FAQ - Kotlin Programming Language. *kotlinlang.org.* [Online] [2020]. [Citace: 28. 12 2020.] https://kotlinlang.org/docs/reference/faq.html#:~:text=Kotlin%20is%20an%20open%2Dsource,release%20was%20in%20February%202016..

28. the Apache Groovy project. The Apache Groovy programming language. ***groovy-lang.org.*** **[Online] 2020. [Citace: 28. 12 2020.] https://groovy-lang.org/.**

29. Core Technologies. ***docs.spring.io.*** **[Online] 2021. [Citace: 14. 3 2021.] https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#spring-core.**

30. @Data. ***projectlombok.org.*** **[Online] 2021. [Citace: 14. 3 2021.] https://projectlombok.org/features/Data.**

31. Spring Framework Overview. ***docs.spring.io.*** **[Online] 2021. [Citace: 14. 3 2021.] https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html#overview.**

32. Tanner Linsley. Overview | React Query | TanStack. ***eact-query.tanstack.com.*** **[Online] 2020. [Citace: 14. 4 2021.] https://react-query.tanstack.com/overview.**

33. traverson/user-guide.markdown at master · traverson/traverson. ***github.com.*** **[Online] GitHub, Inc, 2021. [Citace: 15. 4 2021.] https://github.com/traverson/traverson/blob/master/user-guide.markdown.**

34. Kelly, M. draft-kelly-json-hal-00 - JSON Hypertext Application Language. ***tools.ietf.org.*** **[Online] 2012. [Citace: 15. 4 2021.] https://tools.ietf.org/html/draft-kelly-json-hal-00.**

35. traverson - npm. ***npmjs.com.*** **[Online] 2021. [Citace: 15. 4 2021.] https://www.npmjs.com/package/traverson.**

36. Fielding, Roy Thomas. Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST). ***ics.uci.edu.*** **[Online] 2000. [Citace: 15. 4 2021.] https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.**

37. Fielding, Roy T. REST APIs must be hypertext-driven » Untangled. ***roy.gbiv.com.*** **[Online] 20. 10 2008. [Citace: 15. 4 2021.] https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.**

38. Maven – Introduction. ***maven.apache.org.*** **[Online] The Apache Software Foundation, 2021. [Citace: 17. 4 2021.] https://maven.apache.org/what-is-maven.html.**

39. Spring Data REST Reference Guide. ***doc.spring.io.*** **[Online] 2021. [Citace: 17. 4 2021.] https://docs.spring.io/spring-data/rest/docs/current/reference/html/#intro-chapter.**

40. Avram, Abel a Marinescu, Floyd. ***Domain-Driven Design.*** **místo neznámé : C4Media Inc., 2006. 978-1-4116-0925-9.**

41. Spring HATEOAS - Reference Documentation. ***docs.spring.io.*** **[Online] 2021. [Citace: 19. 4 2021.] https://docs.spring.io/spring-hateoas/docs/current/reference/html/#preface.**

42. Spring Security Reference. ***docs.spring.io.*** **[Online] 2021. [Citace: 19. 4 2021.] https://docs.spring.io/spring-security/site/docs/5.4.6/reference/html5/.**

43. Spring Cloud. ***spring.io.*** **[Online] VMware, Inc, 2021. [Citace: 19. 4 2021.] https://spring.io/projects/spring-cloud.**

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Stanislav Čapek**
Studium:	I1800482
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Docházkový systém**
Název bakalářské práce AJ:	Attendance system

Cíl, metody, literatura, předpoklady:

Cíl práce: Navrhnout a implementovat s využitím aktuálních nastrojů aplikaci Docházkový systém a popsat základní principy použitých technologií.

Struktura práce:

1. Úvod
2. Problematika docházkových systémů
3. Analýza a návrh aplikace
    1. Legislativní rámec
    2. Analýza požadavků na docházkový systém
4. Výběr a popis technologií
    1. Přehled výběru dostupných technologii pro tvorbu klienta
    2. Přehled výběru dostupných technologii pro tvorbu serveru
5. Vybrané aspekty implementace
6. Výsledky a závěr

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. Mgr. Tomáš Kozel, Ph.D.
Datum zadání závěrečné práce:	7.8.2020