FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Systém pro evidenci projektových aktivit
Student:	Martin Ouředník
Vedoucí:	Ing. David Bernhauer
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2020/21

Pokyny pro vypracování

Navrhněte a implementujte systém evidence pracovních aktivit dle požadavků zadavatele.
Proveďte rešerši a pro implementaci využijte architekturu mikroslužeb.
Implementujte služby pro správu uživatelů, evidenci práce na projektech a případné pomocné služby.
Vzhledem k citlivé povaze dat a nařízení GDPR navrhněte vhodné prvky zabezpečení (autentizace a autorizace).
Zdokumentujte navrženou architekturu.
Proveďte uživatelské testování výsledné aplikace, zdokumentujte jej a opravte nalezené chyby.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 18. září 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Systém pro evidenci projektových aktivit

Martin Ouředník

Katedra softwarového inženýrství
Vedoucí práce: Ing. David Bernhauer

30. července 2020

Poděkování

Chtěl bych tímto poděkovat svému vedoucímu práce Ing. Davidu Bernhauerovi za pohotové rady a čas, jež mi věnoval na hodinách konzultací. Dále bych chtěl poděkovat své rodině, přátelům a kolegům, kteří mě podporovali během celého mého prodlouženého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 30. července 2020 . . . . . .

Odkaz na tuto práci

Ouředník, Martin. *Systém pro evidenci projektových aktivit.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce se zaměřuje na systém pro evidenci projektových aktivit ve firmě za využití mikroslužeb. Popisuje v čem má být GraphQL náhradou architektury REST a porovnává jejich výhody. Dále rozebírá vlastnosti a použité technologie již existujících řešení systémů pro monitoring firemních aktivit. Součástí práce je, vzhledem k citlivé povaze dat a nařízení GDPR, rešerše a řešení zabezpečení. Cílem je prototyp systému pro evidenci projektových aktivit, obsahující služby pro evidenci uživatelů, práce na projektech a docházky. Na základě požadavků a konzultací bylo pro řešení tohoto systému použito PHP frameworku Lumen.

Klíčová slova PHP, Laravel, Lumen, Mikroslužby, Evidence, Správa aktivit, GDPR

Abstract

This work focuses on system for managment of project activities in company based on microservices. It shows how GraphQL is supposed to replace architecture REST and compares their advantages. Then concentrates on properties and technologies of existing solutions for monitoring work activities. Part of the work, considering the nature of contained data and General Data Protection Regulation, is focused on research on security solutions. The main goal of this work is to create functioning prototype of system for managing project activities providing attendance tracking, user management and project administration. Based on requirements and meetings PHP framework Lumen was chosen as the final sulution for the prototype.

Keywords PHP, Laravel, Lumen, Microservices, Evidence, Activity monitor, GDPR

Obsah

Seznam obrázků

Seznam ukázek kódu

Úvod

Téměř každý se v dnešní době setkává s různými druhy systémů pro evidenci práce na úkolech. Ve spoustě případů nastává problém, že v zásadě neexistuje systém, který by uměl vše, co potřebuje, nebo naopak má spoustu pro vás zbytečných funkcionalit, které nevyužije. Ve výsledku používá 4 rozdílné služby, které perfektně pokrývají jeho potřeby, problém však nastává s propojením dat mezi sebou. Z tohoto důvodu je cílem práce vytvořit systém, pokrývající veškeré požadavky moderního projektového řízení za pomocí mikroslužeb.

Mikroslužby jsou totiž malé nezávislé aplikace, které vždy zajišťují jednu funkcionalitu, a je tedy jednoduché funkcionality přidávat či odebírat, aniž by se změnil základ systému. Aplikace založené na architektuře mikroslužeb se také dají nezávisle škálovat, a tudíž nestojí v cestě dalšímu rozvoji.

Cíl práce

Cílem této práce je vytvoření prototypu systému pro evidenci projektových aktivit, rešerše existujících řešení a aplikací založených na architektuře mikroslužeb. Současně se práce zabývá zabezpečením veškerých dat a komunikace. Vzniklý prototyp má za cíl umožňovat uživateli sledovat docházku, spravovat projekty a mít přehled nad uživateli. Díky spojení těchto funkcionalit by tato aplikace měla vyhovovat požadavkům především menších firem.

Pro tento prototyp je dle nefunkčních požadavků zadavatele zvolen PHP micro-framework Lumen, jenž vychází z frameworku Laravel. Z tohoto důvodu je použit Laravel jako framework pro frontendovou část aplikace.

Struktura práce

První část práce se zabývá architekturou mikroslužeb, jejich výhodami, nevýhodami a osvědčenými postupy jak architekturu implementovat. Dále se ka-

pitola zaměřuje na případové studie velkých firem, kterým užití architektury mikroslužeb prospělo. Konec je věnován vlastnostem a výhodám technologií REST a GraphQL, které se využívají pro komunikaci mikroslužeb.

Tématem druhé kapitoly je specifikace výstupního prototypu, rešerše existujících řešení systémů pro evidenci projektových aktivit. Dále se kapitola věnuje analýze požadavků zadavatele, kategorií uživatelů a případů užití.

Třetí kapitola popisuje implementaci samotného řešení prototypu systému pro evidenci projektových aktivit. Rozebírá zvolené technologie, jejich klady a zápory, vztahy mezi jednotlivými službami a veškeré funkcionality prototypu. Součástí návrhu bude také zabezpečení aplikace a anonymizace dat z důvodu nařízení GDPR.

Poslední část je tvořena popisem výsledného prototypu a uživatelským testováním reálnými uživateli dle scénářů.

Kapitola 1

Mikroslužby

Tato kapitola se věnuje přiblížení architektury mikroslužeb. Vysvětlí výhody této architektury, ale také její nevýhody dle [1] a [2]. Dále je uvedeno několik osvědčených postupů při implementaci systému založeném na mikroslužbách a v neposlední řadě se zaměří na pár příkladů již existujících řešení využívajících architektury mikroslužeb, se kterými se člověk běžně setká a přiblíží důvod, který je vedl k využití této architektury.

1.1 Architektura mikroslužeb

Přístup mikroslužeb je poměrně nový a umožňuje větší volnost při rozvoji aplikací. Aplikace na bázi architektury mikroslužeb jsou v podstatě aplikace složené ze skupiny malých aplikací, takzvaných mikroslužeb. Mikroslužby jsou totiž nezávislé aplikace, které se dají nezávisle nasadit, testovat, škálovat a hlavně vyvíjet. Každá z těchto mikroslužeb má za odpovědnost správu jedné funkcionality, například správy uživatelů, docházky či projektů. Mikroslužby mezi sebou většinou komunikují za pomocí nějakého jednoduchého mechanismu, jako je například REST (viz 1.3.1), která využívá standardních metod HTTP[2].

1.1.1 Výhody architektury mikroslužeb

Tento přístup dává vývojářům určitou nezávislost, jelikož se správa jednotlivých mikroslužeb může rozdělit na týmy a ty mohou samy dělat technická rozhodnutí a to aniž by ovlivnily ostatní funkcionality výsledného systému. Díky tomu je možné pro jednotlivé služby použít rozdílné jazyky, frameworky nebo technologie a přitom nijak neovlivňovat ostatní mikroslužby. Dále umožňuje výměnu jedné služby za nový model, či naškálování služby pokud služba

[2]HTTP neboli Hypertext Transfer Protocol je internetový protokol sloužící pro komunikaci mezi serverem a prohlížečem či několika servery. HTTP funguje na bázi dotazu a odpovědi, neboli na zaslaný dotaz na server se dostává nějaké odpovědi.

přestane zvládat odpovídat na dotazy dostatečně rychle. Naškálování totiž vytvoří repliku mikroslužby, rozloží množství dotazů na kopii, a tím se dosáhne snížení zatížení.

Další výhodou architektury mikroslužeb je její přehlednost. Veškerá funkcionalita aplikace je rozdělena na aplikace dle jejich úlohy, a tudíž je jednodušší pro porozumění činnosti výsledné aplikace.

Mikroslužby také umožňují flexibilitu řešení, a tak nezabraňují vývoji nebo testování nových technologií. Přístup mikroslužeb výrazně zmenší množství závislostí v projektu, čímž je jednodušší nějaké změny zvrátit.

1.1.2 Nevýhody architektury mikroslužeb

Mezi hlavní nevýhody architektury mikroslužeb patří složitost komunikace mezi jednotlivými službami a bezpečnost komunikace. Každá služba musí ověřit, zda požadavek nepřišel od nežádoucího zdroje, který by k těmto informacím neměl mít přístup. Toto se řeší za pomoci vytvoření mikroslužby Gateway viz kapitola 1.2.1.

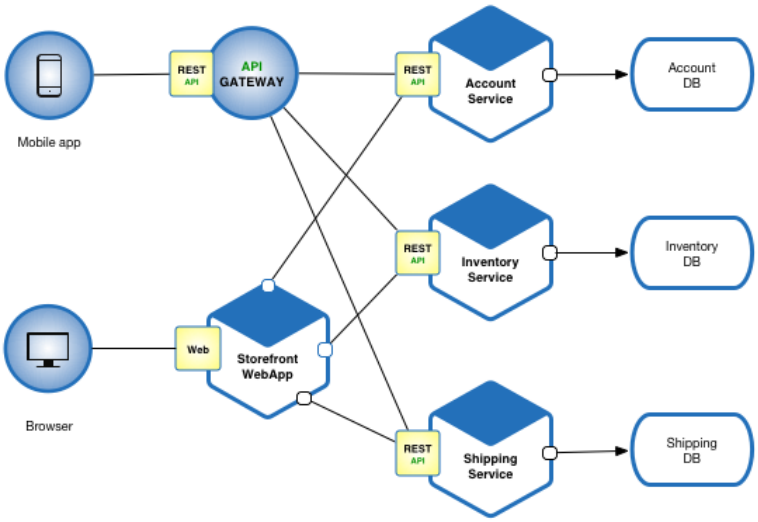
Další nevýhodou, vzhledem ke složité komunikaci, je náročný vývoj aplikace. Aplikace založená na architektuře mikroslužeb vyžaduje zkušené analytiky, následné naplánování a rozdělení celkové aplikace na mikroslužby. Špatná analýza a rozdělení aplikace může vést k úplnému zrušení předchozího řešení a tvorbu nového od začátku.

1.2 Mikroslužby v praxi

Na architektuře mikroslužeb je v dnešní době postaveno velké množství systémů či aplikací. V průběhu tudíž vznikly postupy, které řeší některé nedostatky této architektury. Existují příklady firem, kterým využití této architektury markantně pomohlo v rozvoji, a díky ní se stále drží mezi špičkou ve svém oboru.

1.2.1 Osvědčené postupy

Vzhledem ke komplexní komunikaci je dle publikace D. Namiot a M. Sneps-Sneppe [3] výhodné používat takzvaný gateway. Gateway je samostatná mikroslužba, která se stará o zpracování všech požadavků a po autentizaci o rozdělení požadavků na ostatní mikroslužby. Tímto způsobem se veškeré mikroslužby skryjí, zároveň se jednoduše vyřeší ověřování a zabezpečení aplikace. Další výhodou použití API Gateway je snížení počtu požadavků na jednotlivé mikroslužby, jelikož dovoluje získat data z více mikroslužeb najednou. Nevýhodou použití této praktiky je vývoj a nasazení další služby, díky čemuž dojde ke zvýšení odezvy na požadavky, neboť požadavek prochází skrze Gateway. Je však důležité zmínit, že vzhledem ke sjednocení a zjednodušení požadavků, je tato odezva ve většině případů zanedbatelná. [4]



Obrázek 1.1: Příklad komunikace mikroslužeb viz [4]

Aplikace založená na architektuře mikroslužeb je tvořená velkým množstvím menších aplikací, což znamená velké množství nasazování a nastavování. Z tohoto důvodu se pro nasazení aplikací používaná například řešení Docker[1]. Toto řešení zajistí automatizaci procesu nasazení a zjednoduší úvodní konfiguraci.

1.2.2 Příklady služeb užívající architekturu mikroslužeb

Se službami využívající architektury mikroslužeb se většina lidí setkává na denní bázi a to aniž by si to uvědomovali. Mikroslužby totiž užívá značný počet velkých firem, jako je například Spotify, Netflix, Wallmart či Amazon. Následující kapitola o případových studiích užití architektury mikroslužeb ve firmách je založena na informacích z [5][6].

Wallmart trpěl problémem, že již druhým rokem po sobě během Black Friday nezvládal 6 milionů návštěv na stránku za minutu, čímž výrazně snížil uživatelský prožitek z návštěvy stránky. Tento problém byl zapříčiněn návrhem webu pro rok 2005, kdy většina uživatelů používala počítače nebo notebooky a pro aplikace se užívala architektura monolitická. Roku 2012 byla struktura změněna na architekturu využívající mikroslužby, která měla za cíl obsloužit až čtyři bilióny připojených uživatelů najednou a přes 25 milionů dostupných aplikací. Přechodem na architekturu mikroslužeb vzrostla dostupnost Wallmartu skoro na 100 procent i během událostí jako je Black Friday či Vánoce a nákupy z mobilních zařízení vzrostly okamžitě o 95 procent.

[1] `www.docker.com`

Další firmou jenž využívá architekturu mikroslužeb je Spotify, které ze zkušeností vědělo, že firmy, plánující rozvoj ve velkém měřítku, vyžadují architekturu jenž jde jednoduše škálovat. Nyní má přes 130 milionů[7] prémiových uživatelů a 286 miliónů[8] aktivních uživatelů měsíčně. Pro takové množství uživatelů je zapotřebí vytvořit systém, jehož jednotlivé součásti jdou samostatně škálovat. Tedy výpadku jedné či více služeb si uživatel ani nemusí všimnout, a tím poskytuje nepřerušovaný uživatelský prožitek.

1.3 Komunikace mezi mikroslužbami

Jak je již zmíněno v předešlé části, systém mikroslužeb mezi sebou musí jistým způsobem komunikovat a posílat data. Data by měla mít určitou formu, aby se vyhnulo nekompatibilitě formátů požadovaných aplikacemi, které by ve výsledku mohlo způsobit nekonzistence dat či v horším případě pád některé ze služeb. Z tohoto důvodu se pro komunikaci používají některé z architektur rozhraní s pevně stanovenými standardy komunikace, jako je například architektura REST či GraphQL.

1.3.1 REST

Representational State Transfer neboli zkratkou REST je architektura navržena pro distribuované systémy. Tuto architekturu v roce 2000 pojmenoval a definoval Roy Fielding v rámci své disertační práce Architectural Styles and the Design of Network-based Software Architectures [9]. Fieldingova definice se výrazně od dnešního pojetí REST mezi vývojáři liší, ale základy zůstávají stejné.

Základní pojem pro architekturu tvoří pojem zdroj. Veškeré pojmenovatelné informace jsou zdrojem, tudíž zdroj může být například objekt v databázi, dokument či obrázek. REST je definován na základě omezení, která veškerá komunikace musí dodržovat. Mezi tato omezení patří oddělení klientské části od serverové, bezstavovost, schopnost ukládat do mezipaměti, systém vrstvení, nepovinný Code-On-Demand a jednotné rozhraní.

První omezení určuje, že systém musí být tvořen klientskou a serverovou částí. Což znamená, že server má zdroje, které klientská část potřebuje zobrazit. Serverová část tudíž obstarává ukládání dat, logiku a funkce, zatímco klientská část se stará pouze o uživatelské rozhraní a zobrazení dat. Díky tomuto omezení je docíleno možností více klientských aplikací pro různé typy zařízení a možnosti jednoduše serverovou část škálovat.

Bezstavovost znamená, že každý dotaz na server musí obsahovat veškerá potřebná data aby byl požadavek vykonán, což znamená, že se na serveru neukládají žádná klientská data.

Schopnost ukládat do mezipaměti pro aplikaci znamená, že každý požadavek musí obsahovat informaci, zda je možné informace ukládat do mezipaměti a následně vracet uložená data bez dotazu na server nebo ne. Ukládáním

informací totiž můžeme zajistit rychlejší odezvu na požadavek, ale pokud se odpověď frekventovaně mění může způsobit vrácení nesprávného výsledku.

Systém vrstvení způsobuje, že klient nemůže vědět, zda je připojen ke konečnému serveru nebo data získává přes jiné aplikace. Díky tomuto je systém možno škálovat či přidávat možné aplikace pro zajištění bezpečnosti, a tím zajistit oddělení služby zajištující bezpečnost a samotné logiky aplikace.

Code-On-Demand je jediným nepovinným požadavkem REST architektury a znamená, že serverová část má možnost poslat klientské části skript za běhu, a tím rozšířit jeho funkcionalitu.

Jednotné rozhraní patří mezi nejzásadnější z omezení, které výrazně odlišuje REST od ostatních architektonických stylů. Jednotné rozhraní odděluje implementaci od rozhraní, a tudíž zjednodušuje interakci natolik, že každý uživatel obeznámený se základy architektury REST se rychle zorientuje, a tím se dá aplikace na architektuře REST jednoduše automatizovat. Jednotné rozhraní má 4 omezení.

První z omezení jednotného rozhraní je identifikace zdroje. Každý zdroj musí být unikátně identifikovatelný pomocí URI (Uniform Resource Identifier), což znamená že každý zdroj musí mít své jednoznačné umístění. Dále platí, že pro udržení kompatibility by se URI neměla změnit ani při změně dat zdroje.

Na druhém místě je manipulace se zdrojem skrze jeho reprezentaci. Toto omezení znamená, že se zdroji nezacházíme napřímo pomocí například SQL příkazů, ale pracujeme se zdroji v neutrálním stavu. Nejběžnějším formátem pro reprezentaci stavu zdroje je JSON, který se posílá v těle HTTP požadavků a odpovědí. Pro úpravu získaných dat je tedy zapotřebí upravit data v získaném JSON a ty zaslat s požadavkem na úpravu zdroje zpátky na server.

Další omezení určuje, že každý požadavek nebo odpověď musí obsahovat veškerá data, aby byl příjemce schopný zprávě porozumět. Každá zpráva tedy musí obsahovat hlavičky o formátu zprávy, dále by mělo být jasné co se se zdrojem má stát. Fielding ve své práci nijak nespecifikoval, že se pro REST má používat pouze HTTP, ale vzhledem k tomu, že Fielding je spoluautorem HTTP se mu REST výrazně podobá, čímž se stal nejčastěji používaným protokolem pro RESTful rozhraní. Nejčastěji tedy dle W3Schools [10] REST pro komunikaci požívá čtyř základních HTTP metod:

POST Sloužící k zaslání dat

GET Získává data ze služby

PUT Úprava informací záznamu daty těla požadavku

DELETE Smazání záznamu ze služby

Posledním omezením pro jednotné rozhraní je HATEOAS (Hypermedia as the Engine of Application State). Toto omezení určuje, že další stavy souvislé

aplikace se dají získat pomocí hyperlinků. To znamená, že pro navigaci v aplikaci by se mělo použít odkazů a není zapotřebí vědět jinou než úvodní URI. [11] [12] [13] [14]

K výhodám architektury REST patří jednoduchá škálovatelnost díky oddělení serveru od klienta a flexibilita aplikace. Jelikož data nejsou vázána na metody a služby, mohou se vracet ve všech požadovaných formátech.

Nevýhodou architektury REST je, že si klient nemůže přesně nadefinovat, která data zrovna potřebuje. Tím dochází buď ke ztrátě některých dat nebo je naopak více dat než dle grafiky využije. V prvním případě, kdy klientovi nějaká data schází, se na ně opět musí dotázat. Například pro získání uživatele a jeho docházky se musí vytvořit jeden požadavek na uživatele a druhý na docházku. V druhém případě klient obdrží více než potřebná data, vzniká tak problém při objemnějších databázích, kdy i pro výpis uživatelů získáme veškerá data uživatelů.

1.3.2 GraphQL

GraphQL je nové řešení pro komunikaci mezi serverovou a klientskou částí. Toto řešení obsahuje nový grafový dotazovací jazyk na úrovni aplikační vrstvy. Původně sloužil pro vnitřní potřeby Facebooku, ale roku 2016 Facebook zveřejnil specifikaci a referenci na implementaci. Od zveřejnění je tedy GraphQL open source na GitHub[2]. Vzhledem k tomu, že GraphQL je pouze jazyk se specifikovaným chováním, je možné jej implementovat v jakémkoli jazyce, který bude překládat dotazy v jazyce GraphQL do jazyka, kterému bude server rozumět, např. SQL.

GraphQL je zamýšleno jako náhrada RESTful řešení a snaží se eliminovat jeho nedostatky. Prvním nedostatkem, kterým REST trpěl byla nemožnost definovat data, které chceme získat. V GraphQL si programátor může přesně určit jaká data chce získat. Toho je docíleno tím, že odpověď s daty kopíruje podobu dotazu na API a není globálně určená serverem viz příklad 1.2 a 1.3. Další výhodou GraphQL oproti REST API je možnost získání dat z více zdrojů jediným dotazem. Například pokud uživatel chce v příkladě 1.1 získat i položku friends, může se na ni dotázat viz 1.2.

Ukázka kódu 1.1: Příklad struktury GraphQL

```
type Starship {
id: ID
name: String
length: Float
}

interface Character {
id: ID
name: String
friends: [Character]
```

[2] github.com/graphql

```
}


type Droid implements Character {
id: ID
name: String
friends: [Character]
primaryFunction: String
}

type Human implements Character {
id: ID
name: String
friends: [Character]
starships: [Starship]
}
```

Ukázka kódu 1.2: Příklad dotazu na GraphQL se strukturou 1.1

```
{
  character(name: "Luke") {
    friends {
        on Human {
                        humanFriend:name
                        starships {
                        starship:name
                        length
                        }
                }
                on Droid {
                        droidFriend:name
                        primaryFunction
                }
        }
 }
}
```

Ukázka kódu 1.3: Odpověd na dotaz 1.2

```
{
  "character": {
    "friends": [{
                "humanFriend": "Han",
                "starships": [{
                        "starship": "Falcon",
                        "length": 34.37
                }]
        },
        {
                "droidFriend": "R2-D2",
                "primaryFunction": "Astromech"
        }]
 }
}
```

Dalším nedostatkem RESTful systémů je velké množství endpointů. GraphQL je uspořádán na základě typového systému na rozdíl od unikátních URI

jako je užito v architektuře REST. Díky tomu není zapotřebí složité dokumentace endpointů a všechny dotazy se mohou směřovat na jeden jediný endpoint.

Neposlední výhodou GraphQL je možnost rozšiřovat jednotlivé entity o nová pole či odebírat zastaralá pole bez potřeby jakkoli upravovat existující dotazy. [15] [16] [17]

Kapitola 2

Analýza

Kapitola analýzy se zaměřuje na bližší specifikaci prototypu systému pro evidenci projektových aktivit. Dále obsahuje rešerši obsahující několik příkladů existujících řešení, které se věnují evidenci projektových aktivit, ať už obstarávají pouze jednu funkcionalitu či kompletní řešení. Poslední dvě sekce kapitoly se věnují veškerým funkčním a nefunkčním požadavkům ze strany zadavatele.

2.1 Specifikace

Prototyp systému pro evidenci projektových aktivit má mít možnost vytvářet uživatele. Uživatelé se mohou sami registrovat nebo mohou být vytvářeni administrátorem systému. Každý uživatel musí mít možnost zobrazit a popřípadě upravit své osobní údaje.

Systém musí umožňovat uživatelům evidovat svou práci na jednotlivých aktivitách, mít přehled o předchozích aktivitách a upravovat je.

Aktivity se musí evidovat k projektům, tudíž systém musí umět jednoduše projekty přidávat, měnit a editovat. Systém musí umožňovat administrátorům přehled času stráveného na jednotlivých projektech nebo samostatných úkolech spadajících pod projekt.

Prototyp musí využít architektury mikroslužeb a být rozdělen na jednotlivé komponenty.

2.2 Rešerše existujících řešení

V současné době existuje velké množství řešení pro evidenci projektových aktivit. Některé se zaměřují na jednotlivé funkce, jako jsou evidence úkolů nebo měření času stráveného na aktivitách, jiné se snaží problém pojmout komplexně a poskytují velké množství funkcí.

Bohužel se vždy najde konkrétní případ firmy, které existující řešení nemusí vyhovovat. Každá firma má své firemní procesy a své potřeby, které od systému pro evidenci projektových aktivit vyžadují. Z tohoto důvodu se ve velkém množství firem využívá několik aplikací zároveň, kde každá umí perfektně svou činnost a firmy nejsou limitovány. V případě potřeby jiné funkcionality nebo nevyhovujícímu cenovému plánu se totiž dá jednotlivá aplikace jednoduše vyměnit za jinou díky funkci exportu dat, kterou většina takových služeb podporuje. Dále se jednotlivé aplikace dají propojit pomocí API a pomocné služby jako je například Zappier[3] či Integromat[4], která automatizuje předání dat, transformuje vytažená data a předá je druhé službě, čímž zajistí zmíněné propojení.

Popis existujících řešení v této kapitole bude dle následující struktury. V první části se u systémů popisují hlavní vlastnosti, které by měl systém pro evidenci projektových aktivit obsahovat. Dále se věnuje vlastnostem, které odlišují systém od konkurence a které mohou být pro některé firmy nebo programátory rozhodujícím faktorem. Druhá část obsahuje přehled zařízení na kterých jednotlivé systémy lze užívat. Poté se zaměřuje zda má systém dostupné API a jaké jsou použité technologie k vývoji. Následuje segment věnující se cenové politice systémů a porovnání jednotlivých verzí. Na závěr rozbor shrnuje případy užití systému pro evidenci projektových aktivit a jeho nevýhody.

2.2.1 Trello

Prvním z příkladů řešení věnující se evidenci projektových aktivit je Trello, které slouží k vytváření a přehledu nad stavem jednotlivých úkolů. V aplikaci se tudíž dají vytvářet projekty, úkoly a seznamy činností pro splnění úkolu.

Trello funguje v rámci nástěnek, což znamená, že pro každý projekt máte oddělenou nástěnku. Tyto nástěnky si můžete nastavit dle vlastních potřeb, nebo na ně můžete použít předvytvořenou šablonu, a to jak z oficiálních stránek či vlastní. Šablony se dají tvořit i pro jednotlivé úkoly, což mnohonásobně zjednodušuje práci a předchází problémům, kde se při vytváření úkolu zapomnělo na dodání podkladů nebo přiřazení člena týmu.

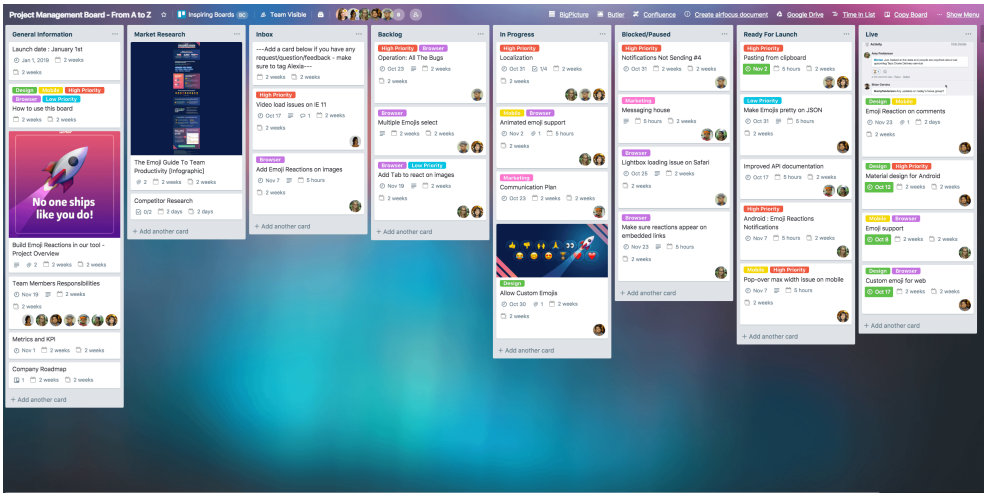
Úkoly mohou mít přiřazené štítky, které se dají použít pro zvýraznění důležitosti jednotlivých zadání, termínů, listů podúkolů, příloh, či dokonce i vlastní pole, která se dají využít pro zmíněnou integraci s dalšími službami.

Trello má od počátku webového klienta, ale vzhledem k dnešní době a velkému využití mobilních zařízení, má i klienty pro Android a iOS. Všichni tito klienti využívají stejné API, které je nyní již z větší části plně odděleno od webového klienta.

Jednou z výhod Trella je jeho API díky němuž se jednoduše dá přistoupit k datům a integrovat s dalšími službami. Další výhodou je přidávání tak-

[3] `zapier.com`

[4] `www.integromat.com`



Obrázek 2.1: Příklad nástěnky Trello z [18]

zvaných vylepšení Trella. Tato vylepšení si může uživatel přidat do svého Trella pro přidání nových funkcionalit jako například přehledy, statistiky, automatizace a měření času na aktivitách. Poslední z užitečných vlastností Trella je Buttler, který slouží pro zjednodušení a automatizaci práce s nástěnkami a kartami. Buttler dokáže přidávat na karty tlačítka, která po kliknutí provedou sérii příkazů, nebo reagují na akce či čas, a tím sníží počet kliknutí pro uživatele.[19]

V poslední době Trello prochází velkým technologickým přesunem klientské části webové aplikace z důvodu přechodu na GraphQL a React[5]. Původně bylo Trello psané v CoffeeScriptu[6] u kterého, vzhledem k rostoucí zákaznické bázi, začali vývojáři dosahovat na limita jeho potenciálu. API Trella využívá technologie REST, tudíž se musela využít služba pro překlad GraphQL požadavků na REST API. Pro databázi Trella je využito MongoDB[7] a pro urychlení fulltextového vyhledávání je použito ElsticSearch[8]. [20] [21]

Trello má tři úrovně platebního plánu ve kterých lze využívat. První úroveň Free je zcela zdarma a přináší veškeré základní funkcionality Trella, ale samozřejmě obnáší různá omezení. Jedno z omezení je na počet týmových nástěnek, kde v tomto plánu je možné mít pouze deset týmových nástěnek. Další je například na velikost nahrávaných souborů, kde je limit omezen na maximální velikost souboru 10 MB. V neposlední řadě je počet vylepšení omezen pouze na jedno a počet příkazů pro Buttlera na padesát měsíčně.

[5] `reactjs.org`
[6] `coffeescript.org`
[7] www.mongodb.com
[8] `www.elastic.co`

Druhá úroveň se nazývá Business Class a platí se 9,99 nebo 12,50 dolarů měsíčně za uživatele. Výše ceny se odvíjí od intervalu platby, je tedy možné si nastavit buď roční či měsíční paušál. V této úrovni již není uživatel omezen počtem nástěnek a limit velikosti souborů je navýšen na 250 MB. Počet vylepšení je na této úrovni neomezen a uživatel může na karty vkládat vlastní pole, hlasování či pokročilé seznamy umožnující přidávat jednotlivým položkám řešitele a termín. Počet příkazů na měsíc pro službu Buttler je také zvýšen na šest tisíc měsíčně. Hlavními vlastnostmi této úrovně je pokročilá správa uživatelů, zabezpečení, prioritní podpora a jednoduchý export dat.

Nejvyšší úroveň Enterprise je cílena převážně na velké firmy a její cenový plán se mění dle počtu uživatelů. Cena se tudíž může pohybovat od 20,83 dolarů za uživatele na měsíc při počtu méně než 300 uživatelů, ale i až na 5,92 dolarů při 5000 uživatelích. Tato úroveň již nemá omezení pro službu Buttler, ale hlavně přináší větší možnosti správy uživatelů, jejich práv na užití nástěnek a další zabezpečení.

Trello již při Free úrovni pokrývá většinu potřeb malých i středních firem pro správu projektů a jednotlivých úkolů, ale postrádá měření stráveného času na aktivitách. To sice lze přidat pomocí vylepšení, ale není natolik intuitivní, postrádá přehledy a statistiky. Hlavní nevýhodou Trella je nemožnost přidání klientů k projektům aniž by se započítali do běžných uživatelů za které se platí. Vzhledem k jednoduchosti a přehlednosti je Trello vhodné pro agilní projektové řízení.

2.2.2 Clockify

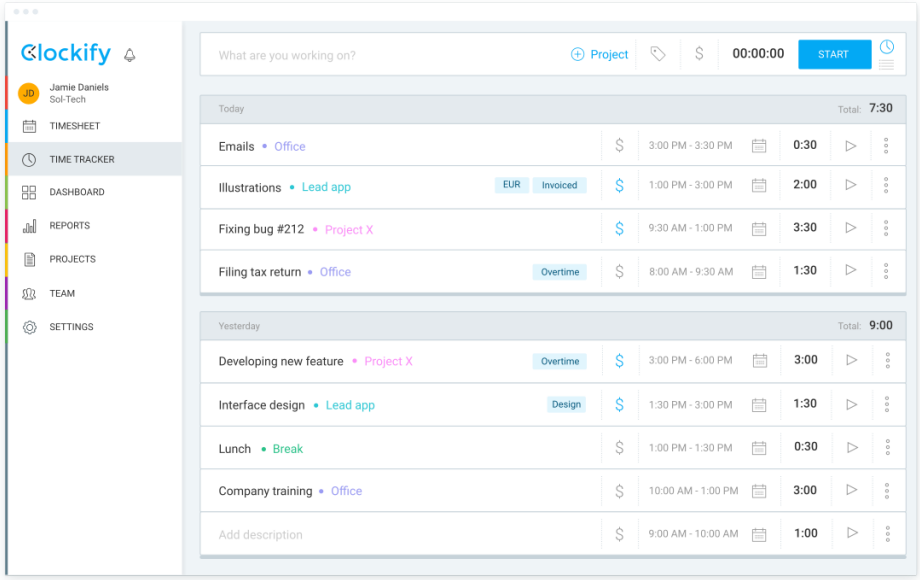
Dalším systémem pro evidenci projektových aktivit je Clockify. Tato aplikace slouží převážně k evidenci času stráveném na jednotlivých aktivitách, ale umožňuje i správu klientů, projektů a úkolů. Výhodou této aplikace zaměřené převážně na měření času strávených na aktivitách je velké množství přehledů, grafů a reportů. Tyto funkcionality jsou perfektní pro klienty, kteří díky nim mají přehled nad prací jednotlivých projektů a tím zjednodušují fakturaci. [22]

Clockify má klientskou aplikaci do mobilních zařízení, webovou verzi, verzi desktopovou, a je možné jej používat i jako rozšíření do vyhledávače Google Chrome či Firefox.[23]

Pro jednoduchou integraci s dalšími službami má Clockify REST API. Počet požadavků je limitován deseti požadavky za sekundu. Clockify má možnost nastavení webhooků, které po provedení vybrané akce pošlou požadavek na zvolenou adresu.[24]

Pro své jednotlivé služby Clockify používá jazyk Java[9] a Node.js[10]. Služby jsou obalené v Docker kontejnerech nastavené službou Consul[11] a spravované

[9] www.java.com
[10] nodejs.org
[11] www.consul.io



Obrázek 2.2: Příklad nástěnky Clockify z [22]

službou Nomad[12]. Klientské verze pro různé platformy jsou psané v jazycích Kotlin, React, Angular a Swift. Pro databázi je využita globálně distribuovaná MongoDB.[25]

Základní verze Clockify je zcela zdarma a obsahuje veškeré základní funkcionality. V pokročilé verzi Plus, která stojí 9,99 dolarů za měsíc bez ohledu na počet uživatelů, je navíc funkce skrývání času pro uživatelské role, zamykaní editace záznamů času a firemní reporty s logem. Dále umožňuje rozšířenou správu nad veškerými záznamy časových aktivit.

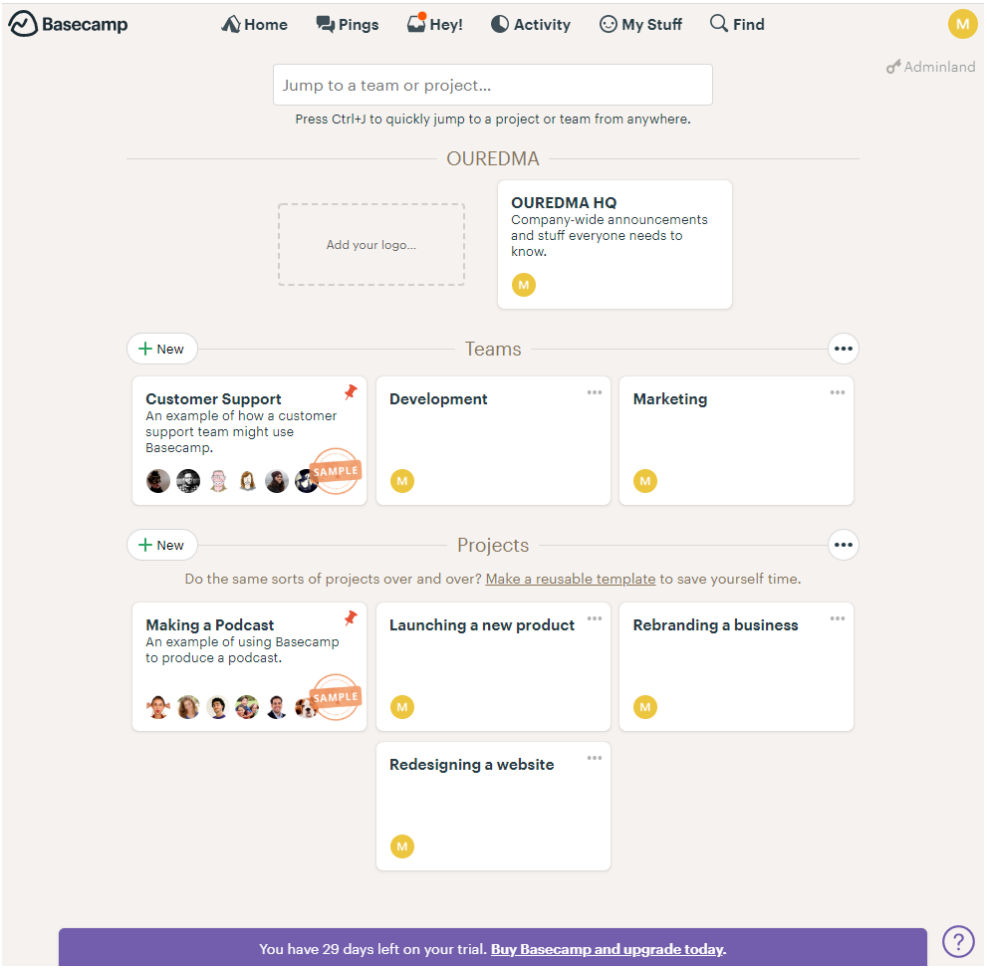
Následuje verze Premium s cenou 29,99 dolarů celkově za měsíc, která oproti předchozí verzi Plus přidává další funkce na lepší správu. Například umožňuje šablony pro projekty, upozornění, hromadné editace, přidávání záznamů pro zaměstnance, skrytí stránek v aplikaci a automatické zamykání editace záznamů.[26]

Clockify je velice všestranná aplikace s propracovaným systémem pro evidenci a přehled stráveného času na jednotlivých úkolech. Jediným nedostatkem aplikace je špatný přehled úkolů a jejich stavů.

[12]www.nomadproject.io

2.2.3 Basecamp

Basecamp slouží k přehledu nad úkoly, které se dají přiřazovat k projektům nebo k týmům. Každý projekt nebo tým má své vlastní úložiště na soubory, nástěnku na zprávy, kalendář či dokonce chatovací místnost. Další vlastností služby Basecamp je možnost vytvořit opakující se otázky pro členy týmu pro neustálý přehled o stavu projektu. Basecamp také u projektů a týmů má možnost vytvořit kalendář, kde se dají zaznamenávat veškerá důležitá data pro projekt, jako je nasazení, testování nebo meetingy.



Obrázek 2.3: Příklad služby Basecamp

Výhodou služby Basecamp je možnost přizvat klienty do projektů, aniž by se sami museli složitě registrovat. Klienti uvidí pouze položky, které se nastaví jako viditelné pro klienty a pokud klienti nechtějí používat Basecamp pro komunikaci je možné zasílat zprávy ze služby na jejich e-mail, načež se

jejich odpověď opět propíše do služby Basecamp, a tím pádem se vyhne nedorozumění a konfliktům v komunikaci jelikož veškeré podklady a komunikace budou přehledné na jednom místě.

Pro užití služby Basecamp stačí prohlížeč, ale má vytvořené klientské verze pro Mac OSx, PC, Android i iOS.

Basecamp je také možné integrovat pomocí nástrojů, jako je již dříve zmíněný Zapier, nebo rovnou integrovat se službami na měření času strávených na jednotlivých úkolech jako například Clockify. Dále pro možnost vlastní integrace má Basecamp dostupné REST API, které je omezeno na padesát dotazů za deset sekund.

Základní verze služby Basecamp je zcela zdarma. Je však limitována pouze na tři projekty, dvacet uživatelů a jeden gigabyte úložiště . Basecamp Business je název hlavního balíčku jenž stojí 99 dolarů celkově. Tento balíček obsahuje neomezený počet projektů, klientů a uživatelů. Dále navyšuje velikost úložiště na 500 GB, přináší prioritní support, veškerou kontrolu nad viditelností obsahu pro klienty, tvorbu týmových projektů a projektové šablony.

Basecamp, se svým zaměřením na obecné pojetí všech potřeb firem a možností jednoduše přizvat klienty do projektů, je ve spoustě případů optimální službou. Jedinými nedostatky jsou chybějící zaznamenávání času stráveném na úkolech a omezené množství stavů u úkolů.

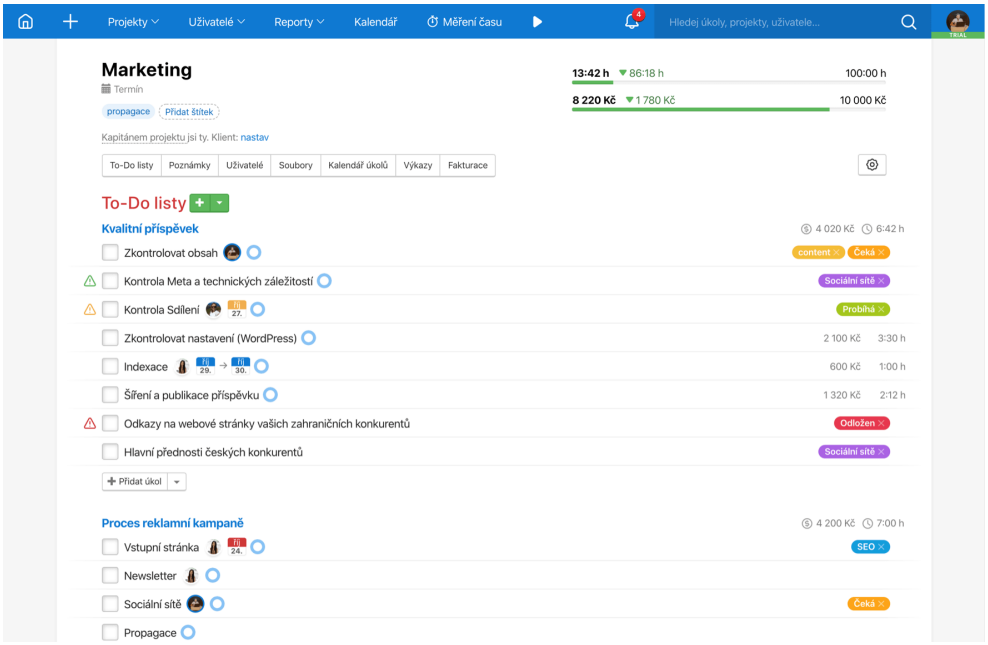
2.2.4 Freelo

V neposlední řadě mezi příklady systémů pro evidenci projektových aktivit patří i služba Freelo, která je vytvořena českou firmou. Freelo zvládá vytvářet projekty se seznamy úkolů a evidovat čas strávený na úkolech. U projektů se dá jednoduše vidět veškerá aktivita na projektu, přidávat poznámky a soubory, přidávat události či termíny do kalendáře nebo fakturovat. Freelo nabízí zdarma šablony projektů na zkopírování do svých účtů, které obsahují například veškeré položky pro testování webových stránek nebo tvorbu nové marketingové kampaně. U jednotlivých úkolů se dá nastavit priorita, na základě které je hned přehledné kolik času se na nich strávilo. Velkou výhodou Freela je napojení na fakturační systémy, jako jsou iDoklad, Fakturoid nebo Vyfakturuj.

Freelo je dostupné v prohlížečích, ale také je možné stáhnout pro mobilní zařízení s operačními systémy Android či iOS.

Jako všechny ostatní zmíněné služby pro evidenci projektových aktivit i Freelo má své REST API, díky němuž se dá integrovat do dalších systémů. Díky integraci se dá jednoduše přejít z nevyhovujícího systému pro evidenci projektových aktivit na Freelo. Toto API je limitováno pouze 25 dotazy na minutu.

Pro backend Freelo zvolilo český framework Nette na jazyce PHP. Dále Freelo užívá mikroslužby, jako je Elastic search pro zrychlení fulltextového



Obrázek 2.4: Nástěnka služby Freelo

vyhledávání nebo RabbitMQ pro zprávy. Klientská verze pro prohlížeče je psaná ve Vue.js.

Základní verze služby Freelo je zdarma a omezuje uživatele na pouze dva projekty, dva uživatele a úložištěm 222 MB.

Další možností je balíček Freelance, který stojí 590 korun českých celkem za měsíc a zvyšuje počet možných projektů na dvacet, odebírá omezení na počet uživatelů, navyšuje velikost úložiště na deset gigabyte a zpřístupňuje API.

Dále je možnost tarifu Team za cenu 990 korun českých měsíčně, jenž oproti Freelance navyšuje počet možných projektů na padesát a zvyšuje velikost úložiště na trojnásobek čili třicet gigabyte. Posledním rozdílem oproti předchozímu tarifu je možnost role správce a projektového manažera pro zlepšení správy systému.

Business je název největšího tarifu, který Freelo nabízí za cenu 2.900 korun českých a odebírá omezení na počet uživatelů. Druhou a poslední změnou oproti předchozímu balíčku Team je rozdíl ve velikosti úložného místa, kde se v Business tarifu navýšil na 100 GB.

Freelo je zdaleka nejzajímavější nástroj pro evidenci projektových aktivit. Působí velice přehledně a díky integracím se systémy pro fakturaci i zásadně urychluje práci. Jediné co Freelo postrádá je zákaznicky přívětivý export dat, jelikož prozatím podporuje pouze export ve formátu CSV, který

si uživatel musí následně upravit. Další výhodou služby Freelo je přívětivý vývojářský tým, který pro budoucí vlastnosti má vytvořené hlasování, kde si každý uživatel může zvolit co by si přál, aby do budoucna Freelo obsahovalo. Díky jednoduché fakturaci, přehlednosti a komunikaci je Freelo výborný systém hlavně pro práci s freelancery.

2.3 Analýza požadavků

Sběr požadavků proběhl po komunikaci se zadavatelem na základě rešerše existujících řešení a po analýze firemních procesů firmy zadavatele. Následující seznamy popisují současné představy o funkčních a nefunkčních požadavcích zadavatele na prototyp.

2.3.1 Funkční požadavky

F1: Registrace uživatele Každý uživatel musí mít možnost se na aplikaci zaregistrovat.

F2: Přihlášení uživatele Pro užití aplikace se musí uživatel přihlásit.

F3: Správa uživatelů Aplikace musí umět zobrazit všechny registrované uživatele a uživatele musí být možno upravovat nebo mazat.

F4: Správa uživatelského profilu Uživatelé mají mít možnost měnit svá uživatelská data.

F5: Smazání uživatelského profilu vzhledem k GDPR Uživatelé se po určité době neaktivity musí sami smazat.

F6: Evidence času na úkolu Uživatelé potřebují evidovat čas strávený na jednotlivých úkolech.¨

F7: Přehled stráveného času na úkolech Každý uživatel musí mít možnost zobrazit svou docházku.

F8: Editace časových záznamů Uživatelé mohou pozměnit své předchozí záznamy o docházce.

F9: Správa projektů Aplikace vyžaduje aby administrátoři měli možnost vypisovat, vytvářet, upravovat a mazat projekty.

F10: Správa úkolů Aplikace administrátory nechá zobrazit přehled jednotlivých úkolů pro projekt a zpřístupní jim jejich správu. Úkoly budou mít možnost nastavit odhadovanou dobu na vypracování.

2.3.2 Nefunkční požadavky

N1: Použití architektury mikroslužeb Jednotlivé služby musí být oddělené a to i API s klientskou částí.

N2: Použití jazyka PHP Pro vytvoření jazyka je důležité užití PHP.

N3: Přístupné API Aplikace musí mít přístupné API pro případnou integraci.

N4: Zabezpečení aplikace Nepřihlášený uživatel se nesmí dostat k datům. Běžný uživatel smí mít přístup jen ke svým datům.

N5: Spolehlivost Aplikace vždy zobrazí aktuální data.

N6: Rozšiřitelnost Možnost jednoduše přidávat další mikroslužby do aplikace.

N7: Škálovatelnost Možnost jednoduše škálovat služby pro možnost budoucího růstu.

2.4 Kategorie uživatelů a případy užití

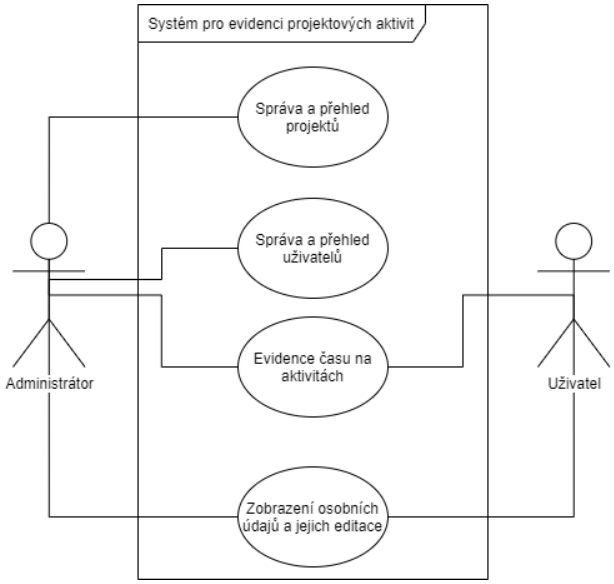
Na základě specifikace a požadavků od zadavatele byly definované dvě uživatelské role a sestaveny případy užití popisující požadované funkcionality systému na základě aktéra. Systém dle požadavků vyžaduje možnost administrátorské role uživatele, který bude mít přehled nad veškerým systémem a možnost běžného uživatele.

Běžný uživatel Běžný uživatel systému pro evidenci projektových aktivit bude mít možnost evidovat svůj strávený čas na jednotlivých úkolech projektů a bude moci upravovat své osobní údaje.

Administrátor Role administrátora systému rozšiřuje běžného uživatele o možnost přehledu a správy projektů. Jednotlivé projekty může mazat, upravovat či přidávat. U každého projektu bude moct spravovat úkoly k zapracování. Administrátor systému dále může zobrazit přehled všech uživatelů a upravovat jejich osobní data. U každého uživatele si administrátor může zobrazit jeho docházku.

2.5 Doménový model

Pro vytvoření požadovaného systému je zapotřebí čtyř entit.



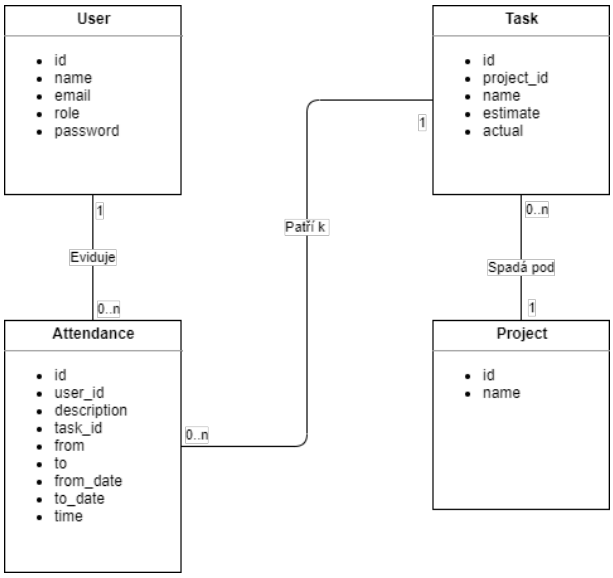
Obrázek 2.5: Diagram případů užití

User Entita uživatelů, která bude mít svůj e-mail a heslo přes které se uživatel bude přihlašovat do aplikace, jméno a roli. Role bude určovat, zda uživatel je administrátorem či běžným uživatelem.

Project Project je entitou projektů u kterých postačuje evidovat jméno projektu.

Task Task slouží jako entita jednotlivých úkolů u kterých je vyžadováno jméno, odhadovaný čas a čas reálný.

Attendance Entita attendance slouží k evidenci jednotlivých časových záznamů o práci na úkolech. Attendance bude mít položku description, kde si uživatelé mohou k jednotlivým záznamům přidávat popisek, časový záznam od kdy do kdy, datum počátku a konce záznamu a čas v sekundách.



Obrázek 2.6: Doménový model

Kapitola 3

Návrh a implementace

3.1 Užité technologie

V této kapitole se práce věnuje všem jazykům, frameworkům či jiným technologiím, které byly využity při implementaci prototypu systému pro evidenci projektových aktivit.

3.1.1 Technologie užité pro Backend

PHP Dle specifikace se zadavatelem a z nefunkčních požadavků 2.3.2 je pro Backend systému užito jazyka PHP (Hypertext Preprocessor), který je jednoduchý a přitom velice flexibilní a výkonný skriptovací jazyk. Tento jazyk je užíván již od roku 1995 a je podporován všemi operačními systémy, jako jsou varianty systémů Unix, Solaris nebo i Windows a Mac OSx [27]. I přes fakt, že PHP není nejrychlejším a nejmodernějším skriptovacím jazykem, tak skoro 80 procent všech webů tento jazyk používá i v roce 2020 [28]. Pro funkčnost systému je zapotřebí verze PHP minimálně 7.2.

Laravel Jako PHP framework je pro klientskou aplikaci užito frameworku Laravel. Hlavní důvody pro volbu tohoto frameworku tvoří jeho rozsáhlá a přehledná dokumentace, jeho početná uživatelská báze a hlavně existence micro-frameworku Lumen. Lumen vychází ze struktury Laravelu a jeho užívání je obdobné. Z tohoto důvodu bylo pro přehlednost a jednoduchost údržby pro jednotlivé mikroslužby zvolen framework Lumen.

Artisan Při práci s výše zmíněnými frameworky je vhodné používat rozhraní v příkazové řádce Artisan. Toto rozhraní umožňuje rychle a jednoduše vytvářet modely, databázové migrace, kontrolery a mnoho dalších. Díky Artisanu a fdatabázovým migracím je nasazení databáze či úprava databázového schématu zcela triviální a vše se dá obstarat pomocí několika příkazů. Artisan

má také možnost vložit různá předem nastavená data do tabulek, a to pomocí funkce seeder, tím je vyřešená i tvorba úvodního administrátorského uživatele.

Composer Pro správu knihoven PHP a jednoduchou instalaci systému je užito utility Composer. Composer lze užívat z příkazové řádky na velkém počtu operačních systémů jako je Linux, Windows či Mac OSx. Composer zajišťuje instalaci veškerých potřebných knihoven a především správných verzí požadovaných knihoven tak, aby nedošlo k nekompatibilitě s verzí PHP či verzí další knihovny.

Guzzle HTTP komunikaci mezi jednotlivými mikroslužbami zařizuje knihovna Guzzle[13]. Guzzle veškerou komunikaci usnadňuje od řešení kódování, přesměrování nebo řešení chyb.

MySQL Vzhledem ke své jednoduché instalaci, velké popularitě použití na webových systémech a volně přístupné licenci je pro databázi zvoleno MySQL. Pro komunikaci s MySQL se jak již z názvu plyne užívá jazyka SQL.

REST API Jeden z nefunkčních požadavků zadavatele viz 2.3.2 tvoří existence přístupného API. Díky využití gateway jsou veškeré mikroslužby dostupné pod jednotným API. Dokumentace endpointů dostupného REST API lze nalézt zde documenter.getpostman.com.

3.1.2 Technologie užité pro Frontend

HTML Pro frontendovou část klientské webové aplikace je užito šablonovacího systému Blade, který je již připravený na práci s Laravelem a je obsažen již v instalačním balíčku Laravelu. Blade používá hlavně jazyka HTML (Hypertext Markup Language), ale nezakazuje užití čistého PHP v šablonách, jelikož se veškeré šablony kompilují do souborů PHP. Jazyk HTML je standardní jazyk používán pro webové stránky a systém Blade je velice přehledný a plný různých funkcionalit, jako je zanořování šablon do šablon, jednoduchý výpis proměnných, podmíněný výpis, iterace polí, autorizace uživatele a jeho role či mnoho dalších.

SASS Pro stylizaci šablon je užito jazyku SASS (Syntactically awesome style sheets), jenž je kompilován do klasického CSS (Cascading Style Sheets). SASS rozšiřuje klasické CSS o spoustu velice užitečných funkcionalit mezi které patří například tvorba proměnných, zanořování stylů, dědičnost stylů a mixiny, které slouží pro vygenerování složitějšího kódu jedním příkazem.

[13] `docs.guzzlephp.org/en/stable/`

Javascript Javascript zařizuje v klientské verzi lepší funkcionality klasických prvků jako jsou HTML `select`, či vstup na zadávání data. Pro tyto účely je využito javascriptových knihoven `moment`, `bootstrap-select` a v neposlední řadě `bootstrap-datepicker`. Knihovna `moment` slouží pro pokročilou práci s s časem a daty a je využita hlavně na stránce s docházkou. Zbylé dvě knihovny `bootstrap-select` a `bootstrap-datepicker` zajišťují uživatelsky přívětivé HTML komponenty s konzistentním vzhledem a chováním skrze všechny prohlížeče.

Laravel Mix Kompilaci javascriptových a SASS souborů zařizuje Laravel Mix, který využívá Webpack. Mix zajišťuje jednoduché nastavení assetů pro build proces Webpacku, a je opět obsažen v základním instalačním balíčku Laravelu. Webpack slouží ke kompilaci zdrojových SASS a javascriptových souborů na produkční. Díky této kompilaci jsou původní soubory spojeny do balíčků a web nemusí zahrnovat velké množství souborů. Mezi další vlastnosti Webpacku patří možná minifikace zdrojového kódu pro zmenšení velikosti a optimalizace SASS kódu pro všechny prohlížeče. Webpack užívá Node.js což je Javascriptový runtime postavený na engine od Google Chrome a zajišťuje běh Javascriptu mimo prohlížeče na serverech či počítačích.

3.1.3 Technologie užité k vývoji

PhpStorm Jako vývojové prostředí byl zvolen PhpStorm od JetBrains, který umožňuje přehled nad celým projektem a jednoduchou interakci se soubory. PhpStorm dále má velké množství funkcionalit, jako je zobrazení databáze, našeptávač při psaní kódu, automatické nahrávání souborů, příkazovou řádku, nebo verzování souborů.

GitKraken Verzovací systém zahrnutý ve vývojovém prostředí sice je dostačující, ale pro lepší přehlednost a lepší kontrolu byl zvolen program GitKraken. GitKraken totiž názorně ukazuje jednotlivé vývojové větve a zjednodušuje používání verzovacího systému. Jak z názvu služby plyne, GitKraken k verzování užívá Git a nabízí možnou integraci se službami jako jsou GitLab, GitHub či Bitbucket, a tudíž zjednodušuje klonování a práci s jednotlivými repozitáři.

Postman Testování dotazů na jednotlivé mikroslužby obstarala aplikace Postman. Postman přináší uživatelsky přívětivé prostředí pro zasílání požadavků na mikroslužby a také přináší možnost opakování dotazů či historie. Další vlastností, jenž byla při vývoji prototypu systému pro evidenci projektových aktivit užita, je tvorba dokumentace API.

Mamp Pro lokální vývojový server bylo užito služby Mamp, která slouží k vytvoření lokální databáze a lokálního serveru. Mamp podporuje tvorbu lokálních serverů na systémech Windows a Mac OSx.

Cloudways Produkční server je hostován na službě Cloudways. Cloudways přináší spoustu nastavení pro usnadnění práce se službami. Jednou z vlastností, které byly použity pro aplikace, je jednoduché zapnutí SSL (Secure Sockets Layer) za pomocí volně přístupného certifikátu od Let's Encrypt. Tento certifikát zajišťuje šifrovanou komunikaci mezi službami. Další vlastností služby Cloudways je nasazení projektu z GIT repozitáře. Díky této vlastnosti je nasazování změn jednoduché, jelikož se změny nasadí při každém příkazu GIT push na repozitář.

3.2 Architektura

Vzhledem k nefunkčním požadavkům zadavatele 2.3.2 je jako architektura systému pro evidenci projektových aktivit zvolena architektura mikroslužeb. Použití této architektury viz 1.1.1 splňuje hned další z nefunkčních požadavků a to snadnou rozšiřitelnost 2.3.2.

3.2.1 Rozdělení aplikace na mikroslužby

Systém pro evidenci projektových aktivit je dle 1.2.1 rozdělen na klientskou webovou aplikaci, mikroslužbu starající se o uživatele, mikroslužbu věnující se projektům a úkolům, mikroslužbu zajišťující veškeré funkcionality pro evidenci docházky a finálně API Gateway.

Webový klient Tato aplikace se převážně stará o uživatelsky přívětivé prostředí pro práci s daty v mikroslužbách. Pro webového klienta je užito frameworku Laravel. Webový klient vykresluje veškerá vyžádaná a následně obdržená data z HTTP požadavků uživatelům. Pro přihlášení do aplikace je odeslán požadavek na User manager, ve kterém se uživatel přihlašuje.

User manager Tato služba slouží k udržení veškerých uživatelských dat a k autentizaci uživatelů pomocí Laravel Passport viz 3.3. Entita user podporuje veškeré CRUD (Create, Read, Update, Delete) operace.

Project manager Project manager je mikroslužba starající se o projekty a jednotlivé úkoly na projektech. Entita Project a entita Task podporují veškeré základní CRUD operace. Entita Task navíc podporuje operaci přidání času k úkolu. Tato operace se užívá například při ukončení aktivity z Attendance manageru pro aktualizování reálně stráveného času na úkolu.

Attendance manager Služba Attendance manager slouží ke správě docházky uživatelů. Služba dokáže získat data o neukončené docházce uživatele nebo veškerou docházku uživatele, zahájit a ukončit stopování aktivity, a také upravit nebo odebrat záznam aktivity.

API Gateway API Gateway je zde využit pro odstínění jednotlivých mikroslužeb za jednotnou službu viz. 1.2.1. API Gateway tedy podporuje veškeré možné operace které jsou dostupné na mikroslužbách. Navíc pro zabezpečení vyžaduje nejdříve uživatelské přihlášení, které je opět zprostředkováno službou User manager.

3.2.2 Komunikace mikroslužeb

Pro komunikaci mikroslužeb bylo zvoleno architektury REST 1.3.1. Tohoto rozhodnutí bylo docíleno po předchozí komunikaci se zadavatelem, který dle nefunkčních požadavků 2.3.2 vyžadoval jednoduchou škálovatelnost, kterou REST viz 1.3.1 nabízí. Dalším důvodem je jednoduchá struktura endpointů vzhledem k jednotnému rozhraní této architektury. Zadavatel je z předchozích projektů se základy architektury REST obeznámen a pro budoucí rozšířitelnost ze strany zadavatele je REST optimálním řešením.

3.3 Zabezpečení

Veškerá komunikace mezi mikroslužbami a data systému na evidenci projektových aktivit musí být zabezpečena. Jednotlivé mikroslužby požadují, aby požadavek přišel se správným tokenem v hlavičce požadavku a aby požadavek přišel ze stejné IP adresy jako je mikroslužba. Tímto se docílí, že ani při zjištění koncových API a posláním požadavku mimo autentizační Gateway či webovou aplikaci, mikroslužba nevrátí žádná citlivá data.

Pro přihlášení do webové aplikace je využito Laravel Passport a Laravel Socialite. Laravel Passport vytváří možnost autentizace přes API díky OAuth2. OAuth2 je standardní protokol autorizace zaměřený na zjednodušení autorizace a zároveň stanovení standardních postupů. Laravel Socialite usnadňuje využití OAuth2 pro registraci a přihlášení do webové aplikace.

Autentizace pro využití REST API Gatewaye opět probíhá za využití OAuth2 protokolu dotazem na autentizační mikroslužbu.

3.3.1 Omezení přístupu běžným a nepřihlášeným uživatelům

Uživatelé systému pro evidenci projektových aktivit mají jednotlivé role. Po registraci je každému uživateli nastavena role user a tuto roli může administrátor systému kdykoli změnit. Při instalaci systému je automaticky vytvořen uživatel root s úvodním heslem, které si po přihlášení může změnit. Jednotlivé akce a funkcionality jsou nadále uživatelskou rolí omezeny pomocí

middleware, který kontroluje, zda přihlášený uživatel má práva pro tuto akci či ne. Nepřihlášený uživatel je vždy ve webové aplikaci pomocí autentizačního middleware přesměrován na přihlášení a má zamítnuté veškeré akce v aplikaci.

3.3.2 Zajištění citlivých dat vzhledem k GDPR

Dle [29] údaje, které nejsou potřebné pro účel pro který byly shromažďovány nebo zpracovávány, je potřeba bez zbytečného odkladu vymazat. Zároveň platí, že zaměstnanec má právo do tří let uplatit nároky vůči zaměstnavateli u soudu [30]. Toto je odvozeno ze tříleté lhůty promlčení dle § 629 odst. 1) Nového Občanského zákoníku [31]. Dále dle § 96 zákona č. 187/2006 Sb. je zaměstnavatel povinen uchovávat záznamy o zaměstnancích účastných pojištění po dobu deseti let[32]. Z tohoto důvodu budou data o docházce uchována po dobu deseti let od poslední aktualizace záznamů. Následně pokud program zjistí, že uživatel neprojevoval žádnou aktivitu v posledních třech letech je automaticky vymazán a jeho veškerá data o docházce a práci na projektech jsou taktéž smazána.

Toto smazání je provedeno Cronem a pomocí Laravel Scheduler. Cron je softwarová utilita, která slouží ke spouštění příkazů v pravidelných intervalech. Cron tedy každý den o půlnoci spustí Laravel Scheduler, který má nadefinovanou funkci na odstranění uživatele jehož záznamy jsou již nepotřebné.

Kapitola 4

Uživatelské rozhraní

4.1 Výsledný prototyp

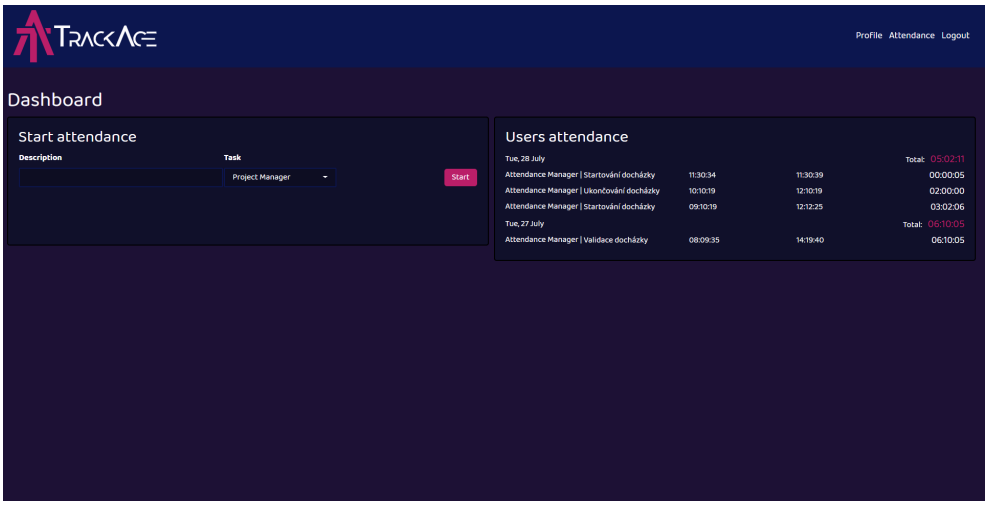
Klientská část prototypu systému pro evidenci projektových aktivit je dostupná na adrese ouredma.cz, API Gateway je přístupný pod adresou gateway.ouredma.cz a zdrojové kódy pro všechny mikroslužby jsou dostupné na službě GitLab.

Stránka přihlášení Přihlášení a registrace do klientské aplikace probíhá jak bylo zmíněno v 3.3 skrze OAuth2 požadavek na autentizační mikroslužbu, neboli User manager. Zde se uživatel registruje či přihlásí a následně je dotázán, zda si přeje aby klientská aplikace získala přístup k jeho účtu. Po schválení je uživatel přesměrován opět na klientskou aplikaci a automaticky přihlášen.

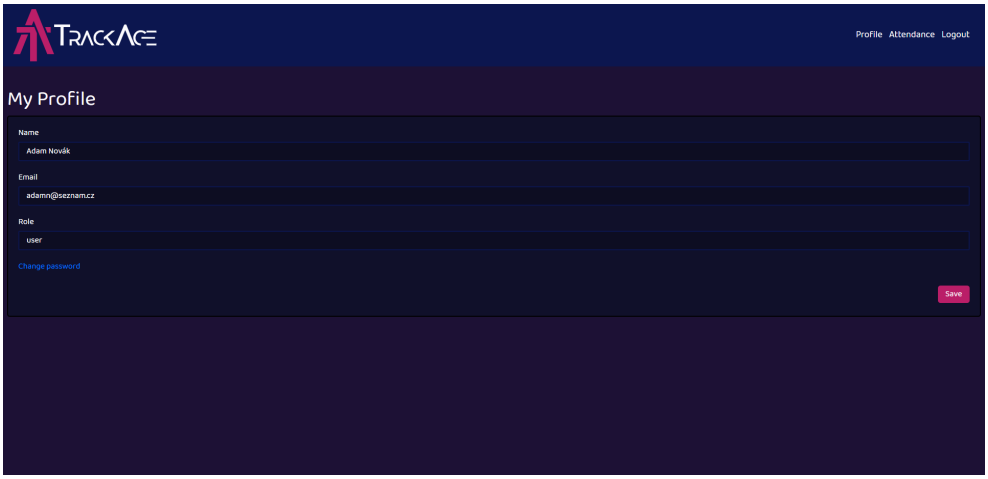
Uživatelská nástěnka Úvodní stránku aplikace tvoří nástěnka. Zde má uživatel přehled posledních záznamů o docházce a také má možnost jednotlivé aktivity stopovat. Stopování aktivit má stejnou funkcionalitu jako na stránce správy docházky.

Stránka detailu uživatele Stránka detailu uživatele zobrazuje veškerá osobní data, která má uživatel možnost měnit. Jediné pole, které jako běžný uživatel nelze změnit, je uživatelská role. Toto chování je z bezpečnostních důvodů ošetřeno.

Stránka správy docházky Tato stránka slouží k veškeré manipulaci se záznamy aktivit na jednotlivých projektech. Hlavní část stránky tvoří část pro stopování aktivit, kde si uživatel může u aktivit vybrat úkol na kterém čas tráví a přidat k němu popisek. Po spuštění stopování času má uživatel přehled nad dobou strávenou na aktivitě aktualizovanou každou sekundu. Dle potřeby může změnit datum a čas počátku měření.

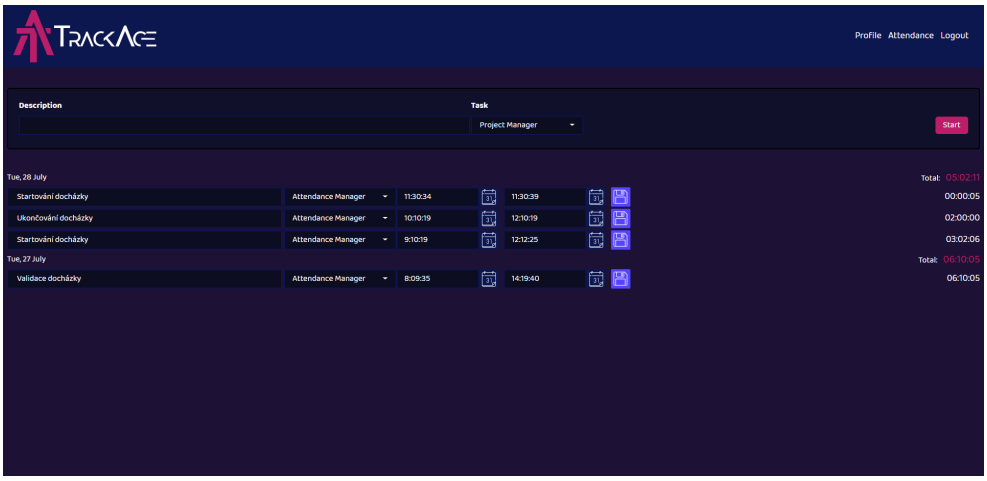


Obrázek 4.1: Uživatelská nástěnka



Obrázek 4.2: Stránka detailu uživatele

Výpis aktivit je seskupen dle data aktivity s přehledem celkového času stráveném na aktivitách za daný den a seřazen od nejaktuálnějších záznamů. Každý záznam má veškeré položky plně editovatelné a po kliknutí na tlačítko uložit se záznam upraví v databázi.



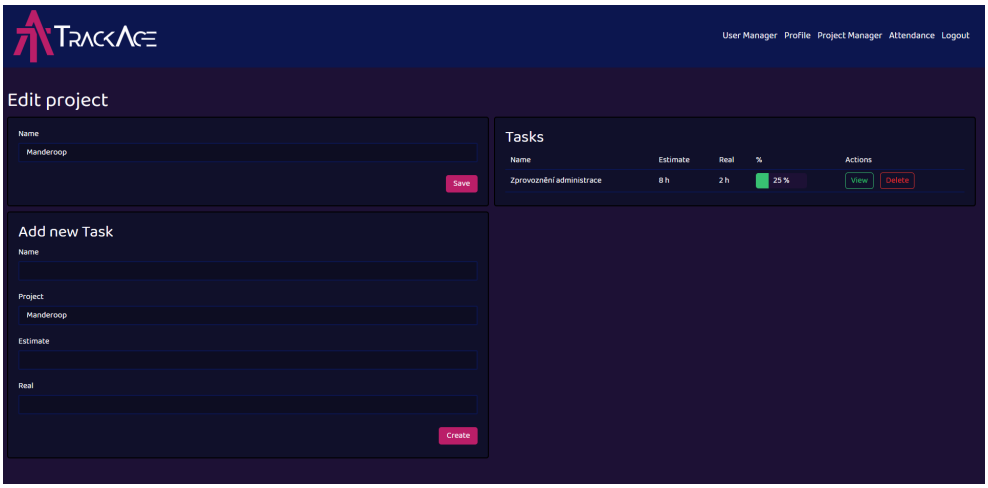
Obrázek 4.3: Stránka správy docházky

Stránka přehledu projektů Stránka s přehledem projektů obsahuje seznam projektů, u kterých je možné zobrazit detail nebo projekt odstranit. Dále obsahuje formulář na přidání nového projektu. Tato stránka je běžným uživatelům nepřístupná.

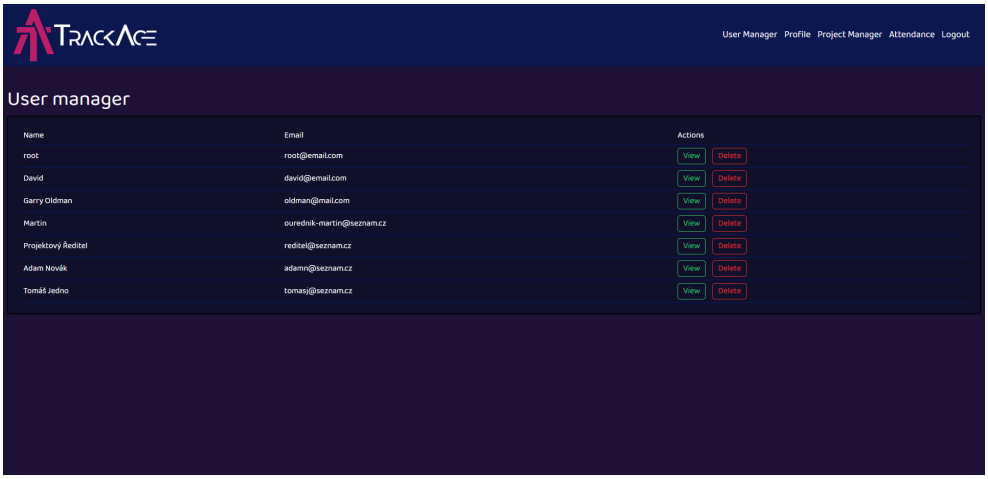
Stránka detailu projektu Detail projektu obsahuje jednoduchou editaci projektových informací, formulář pro přidání nového úkolu a přehled projektových úkolů. Jednotlivé projekty mají u sebe přehled odhadovaného času na splnění projektu a reálného času, jenž je na projektu strávený. Poměr těchto položek je pro lepší přehled graficky znázorněn. Úkoly se dají nadále rozkliknout pro zobrazení detailu úkolu, nebo se dají smazat. Stránku detailu projektu mohou zobrazit jen uživatelé s rolí admin.

Stránka detailu úkolu Tato stránka obsahuje jednoduchou editaci veškerých informací o úkolech a je přístupná pouze administrátorovi systému.

Stránka přehledu uživatelů Stránka přehledu uživatelů obsahuje seznam veškerých uživatelů systému pro evidenci projektových aktivit. Tato stránka je dostupná pouze uživatelům s administrátorskou rolí v systému. Tento uživatel nadále může zobrazovat a upravovat osobní údaje jednotlivých uživatelů či uživatele odstraňovat.



Obrázek 4.4: Stránka detailu projektu



Obrázek 4.5: Stránka přehledu uživatelů

4.2 Uživatelské testování

Z rešerše stávajících řešení systémů pro evidenci projektových aktivit byly do prototypu implementovány funkcionality, které byly zhodnoceny jako užitečné a uživatelsky přívětivé. Pro otestování, zda tato rozhodnutí byla správná, bylo zapotřebí klientskou verzi aplikace otestovat reálnými uživateli. Testování proběhlo na třech uživatelích, kde každý uživatel zastupoval jednu z následujících skupin, které budou systém využívat. Nezastoupenou skupinou jsou vlastníci startupu, jejichž scénář se prolíná se scénáři ostatních uživatelských skupin.

4.2.1 Skupiny uživatelů

První skupinu uživatelů tvoří projektoví ředitelé firem. Tento uživatel potřebuje vytvořit pro svůj tým úkoly, které bude shlukovat pod jednotlivé projekty. Dále takový uživatel potřebuje možnost jednotlivé členy týmu spravovat a mít přehled o jejich stráveném čase na úkolech.

Další skupinou je profesionál na volné noze neboli freelancer. Tento uživatel chce mít přehled kolik času stráví na jednotlivých úkolech pro své klienty a mít tak pro ně podložené údaje o práci na projektu. Freelancer si povede seznam klientů nebo projektů, u kterých si bude evidovat různé úkoly. Bude využívat funkcionalitu sledování času a zasílat přehledy záznamů.

Následující skupinou je zaměstnanec v softwarové firmě, kde je potřeba sledovat kolik hodin z pracovní doby tráví přímo na projektu a kolik věnuje jiné činnosti. Takový uživatel bude hlavně využívat možnost stopování času stráveném na úkolech.

Poslední skupinou jsou vlastníci startupu. Ti mají většinou jen pár členů týmu, převážně na dohodu o provedení práce, potřebují tak sledovat kolik času měsíčně členové týmu stráví prací, aby měli potřebné podklady pro tvorbu mzdy. Vlastník může kdykoli odebírat a měnit uživatele, protože se v takových podmínkách tým frekventovaně mění. Sám si také může sledovat kolik času stráví nad prací pro svou firmu a může organizovat projekty a jednotlivé úkoly spadající pod projekt.

4.2.2 Testované scénáře

Na základě zmíněných skupin uživatelů byly sestaveny následující scénáře, které se následně budou testovat s reálnými uživateli.

První scénář bude využíván především skupinou projektových ředitelů, nebo vlastníků startupů. Prvně se tento uživatel přihlásí s administrátorským účtem do aplikace. Dále zobrazí přehled uživatelů, vybere uživatele *Adam Novák* a tomu zkontroluje docházku. Poté vytvoří nový projekt s názvem *Mongobongo* a do něj přidá úkol optimalizace načítání. Následně zobrazí přehled

všech projektů, otevře si projekt s názvem *Ouredma*, který přejmenuje na *TrackAge* a úkol *User manager* smaže. Nakonec se uživatel odhlásí.

Scénář číslo dvě bude odpovídat využití aplikace profesionálem na volné noze. Ten se přihlásí do aplikace účtem s právy administrátora. Vytvoří si nový projekt se jménem *Manderup*, ke kterému si přidá úkol *zprovoznění registrace* s odhadovaným časem *1 hodina*. Následně půjde do části s evidencí docházky a zapne si sledování času na vytvořený úkol. Dále upraví u úkolu odhadovaný čas na *12 hodin* a ukončí sledování času. Záznam o práci na úkolu upraví změnou data a času spuštění na poledne předchozího dne a následně se odhlásí.

Třetí scénář je relevantní pro zaměstnance v softwarové firmě. Tento uživatel se přihlásí a spustí sledovaní času se zvoleným úkolem *TrackAge*. Následně upraví své uživatelské jméno na *Moritz Straube*. Sledování času ukončí. U vytvořeného úkolu změní počátek o hodinu dříve a přejde na nástěnku. V poslední řadě se uživatel odhlásí.

4.2.3 Vyhodnocení

Po testování prvního scénáře vyplynulo, že položky v menu nejsou seřazené dle užitečnosti. Položku profilu a přehled uživatelů by testovaný subjekt přesunul v menu napravo, a naopak přehled projektů a docházky by dal na první pozice. Dále uživateli nepřišlo přívětivé barevné schéma systému a měl problém s editací projektu. Do budoucna by uživatel uvítal přehled stráveného času uživatelů na projektu ve volitelném časovém období.

Z testování scénáře odpovídajícího profesionálovi na volné noze vznikla připomínka na doplnění jednotlivých položek v menu o ikony pro zrychlení orientace v aplikaci. V tomto případě uživatel taktéž zmínil potřebu změny pořadí položek v menu. Dále na nástěnce uživatele chybí názvy jednotlivých sloupců při výpisu docházky a název *Users attendance* je zavádějící. Dále u úkolů chybí ke kterému projektu patří a v případě dvou stejných úkolů u dvou projektů dochází ke zmatení uživatele. V poslední řadě by uživatel do budoucna uvítal možnost plánování úkolů do kalendáře a možnost vypnutí funkcionality *User managera*, jelikož uživatel této skupiny ji nevyužije.

Poslední scénář věnující se užití zaměstnancem v softwarové firmě objevil špatné zařazení časového záznamu při evidenci docházky v rámci několika dní. I v tomto případě uživateli přišlo barevné schéma systému nepřehledné a ocenil by ikony pro ilustraci položek v menu. Do budoucna by tento uživatel ocenil možnost zobrazení docházky pro jednotlivá časová období.

Finálně všichni uživatelé ocenili rychlost a jednoduchost systému a scénáře zvládli vykonat bez dlouhého hledání. Přehlednost textů a rozdělení na bloky přispívají k jednoduché orientaci. Dle připomínek bylo barevné schéma práce přepracováno, do menu byly k jednotlivých položkám přidány ikony, byla přidána legenda na nástěnce uživatele u přehledu docházky a funkcionalita editace projektu byla také opravena. Název na domovské stránce s přehledem docházky byl upraven dle požadavků a výpis evidence docházky byl přepra-

cován pro záznamy překračující jeden den. U jednotlivých úkolů byla pro větší přehlednost přidána i zkratka projektu.

Závěr

Cílem této práce bylo navrhnout a vytvořit prototyp systému pro evidenci projektových aktivit na bázi mikroslužeb. Dalším z cílů práce bylo provést uživatelské testování výsledného prototypu a demonstrovat jednotlivé případy použití na reálných uživatelích.

V první části v rámci rešerše byly zjištěny ověřené postupy při tvorbě aplikací na bázi mikroslužeb a užitečné vlastnosti již existujících řešení systémů pro evidenci projektových aktivit. Poté se rozebraly možné způsoby komunikace mezi jednotlivými mikroslužbami, porovnaly se jejich rozdíly a na základě těchto zjištění a rešerše existujících řešení byla vybrána architektura REST.

Implementace prototypu se odvíjela na základě informací získaných z analýzy a požadavků zadavatele. Byly vytvořeny jednotlivé mikroslužby obstarávající veškeré požadované funkcionality. Všechny mikroslužby byly následně nasazeny a jsou dostupné skrze webového klienta na adrese ouredma.cz nebo za pomocí API Gateway dostupného z gateway.ouredma.cz. Zdrojové kódy jsou veřejně dostupné na GitLab repozitáři.

Finálně proběhlo uživatelské testování systému pro evidenci projektových aktivit reálnými uživateli, kterým byly zadány různorodé úkony v aplikaci. Na základě výsledků testování byl vyhodnocený uživatelský zážitek ve webovém klientu. Nalezené chyby a nedostatky byly opraveny.

Cíle práce byly splněny. Oproti původnímu zadání byl systém rozšířen o jednotlivé úkoly a API Gateway pro možnou budoucí integraci s dalšími službami. Vzhledem k tomu, že výsledkem je pouze prototyp bylo by do budoucna vhodné například přidat uživatelům na nástěnku přehled jednotlivých úkolů či rozšířit administraci dílčích úkolů o stav, ve kterém se nacházejí. Dále by se systém dal rozšířit o mikroslužbu na export dat, či možnou mikroslužbu pro komunikaci v týmech.

Seznam použité literatury

1. TAIBI, Davide; LENARDUZZI, Valentina; PAHL, Claus; JANES, Andrea. *Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages*. 2017. Dostupné z DOI: 10.1145/3120459.3120483.
2. NEWMAN, S. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015. ISBN 9781491950333. Dostupné také z: https://books.google.cz/books?id=jjl4BgAAQBAJ.
3. NAMIOT, Dmitry; SNEPS-SNEPPE, Manfred. On Micro-services Architecture. *Interenational Journal of Open Information Technologies*. 2014, roč. 2, s. 24–27.
4. RICHARDSON, Chris. What are microservices? *Microservices.io* [online]. 2017 [cit. 2020-06-20]. Dostupné z: https://microservices.io/.
5. NOVOSELTSEVA, EKATERINA. Benefits & examples of Microservices architecture. *Apiumhub* [online]. 2018 [cit. 2020-05-30]. Dostupné z: https://apiumhub.com/tech-blog-barcelona/microservices-architecture-implementation/.
6. MAGUIRE, Jamie. Microservice Architecture (Examples and Diagram). *DevTeam.Space* [online]. 2020 [cit. 2020-05-30]. Dostupné z: https://www.devteam.space/blog/microservice-architecture-examples-and-diagram/.
7. WATSON, Amy. *Spotify users - subscribers in 2020* [online]. 2020 [cit. 2020-05-30]. Dostupné z: https://www.statista.com/statistics/244995/number-of-paying-spotify-subscribers/.
8. WATSON, Amy. *Spotify MAUs worldwide 2019* [online]. 2020 [cit. 2020-05-30]. Dostupné z: https://www.statista.com/statistics/367739/spotify-global-mau/.

9. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Dostupné také z: `https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`. Disertační práce. University of California, Irvine.

10. W3SCHOOLS.IN. *REST Methods* [online]. 2020 [cit. 2020-07-14]. Dostupné z: `https://www.w3schools.in/restful-web-services/rest-methods/`.

11. ERL, Thomas; CARLYLE, Benjamin; PAUTASSO, Cesare; BALASUBRAMANIAN, Raj. SOA with REST - Principles, Patterns and Constraints for Building Enterprise Solutions with REST. In: *The Prentice Hall service technology series*. 2012.

12. LANGE, Kenneth. What are RESTful Web Services? *Kenneth Lange* [online]. 2016 [cit. 2020-06-27]. Dostupné z: `https://www.kennethlange.com/what-are-restful-web-services/`.

13. HORDĚJČUK, Vojta. REST. *Vojta Hordějčuk* [online]. 2017 [cit. 2020-06-27]. Dostupné z: `http://voho.eu/wiki/rest/`.

14. RESTFULAPI.NET. REST API Tutorial. *What is REST – Learn to create timeless REST APIs* [online]. 2018 [cit. 2020-06-27]. Dostupné z: `https://restfulapi.net/`.

15. BUNA, Samer. *Learning GraphQL and Relay*. Birmingham, United Kingdom: Packt Publishing, 2016. ISBN 9781786465757.

16. NAUTRON.CO. *GraphQL dotazovací jazyk pro vaše moderní API* [online]. 2018 [cit. 2020-07-06]. Dostupné z: `http://graphql.cz/graphql-dotazovaci-jazyk`.

17. HARTIG, Olaf; PÉREZ, Jorge. Semantics and Complexity of GraphQL. In: *Proceedings of the 2018 World Wide Web Conference*. Lyon, France: International World Wide Web Conferences Steering Committee, 2018. WWW '18. ISBN 9781450356398. Dostupné z DOI: `10.1145/3178876.3186014`.

18. ALEXIA. How To Become A Project Management Master With Trello. *Trello Blog* [online]. 2018 [cit. 2020-06-11]. Dostupné z: `https://blog.trello.com/project-management-power-ups`.

19. TRELLO. *An Introduction to Butler* [online]. 2019 [cit. 2020-06-26]. Dostupné z: `https://help.trello.com`.

20. CLARK, Barry. Tech at Trello. *The Trello Tech Stack 2016* [online]. 2016 [cit. 2020-06-11]. Dostupné z: `https://tech.trello.com/the-trello-tech-stack/`.

22. INC., COING. *The only truly free time tracking software* [online]. 2020 [cit. 2020-07-06]. Dostupné z: `https://clockify.me/feature-list`.

23. INC., COING. *Time tracking apps* [online]. 2020 [cit. 2020-07-06]. Dostupné z: `https://clockify.me/apps`.

24. INC., COING. *Clockify API* [online]. 2020 [cit. 2020-07-06]. Dostupné z: `https://clockify.me/developers-api`.

25. KANDIC, Jovana [private communication]. 2020. Cesta: clockify.eml.

26. INC., COING. *Extra Features* [online]. 2020 [cit. 2020-07-06]. Dostupné z: `https://clockify.me/extra-features`.

27. GROUP, PHP. *platforms* [online]. 2017 [cit. 2019-07-14]. Dostupné z: `https://wiki.php.net/platforms`.

28. Q-SUCCESS. *Usage statistics of PHP for websites* [online]. 2020 [cit. 2019-07-14]. Dostupné z: `https://w3techs.com/technologies/details/pl-php`.

29. ŠKORNIČKOVÁ, Eva. *Právo na výmaz* [online]. 2020 [cit. 2020-07-20]. Dostupné z: `https://www.gdpr.cz/gdpr/heslo/pravo-na-vymaz`.

31. PODNIKATEL.CZ. *Nový Občanský zákoník* [online]. 2020 [cit. 2020-07-20]. Dostupné z: `https://www.podnikatel.cz/zakony/novy-obcansky-zakonik/f4580932/`.

Příloha A

Seznam použitých zkratek

API Application Programming Interface.

CRUD Create, Read, Update, Delete.

CSS Cascading Style Sheets.

GB Gigabyte.

GDPR General Data Protection Regulation.

HATEOAS Hypermedia as the Engine of Application State.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

JSON JavaScript Object Notation.

MB Megabyte.

PHP Hypertext Preprocessor.

REST Representational State Transfer.

SASS Syntactically awesome style sheets.

SQL Structured Query Language.

SSL Secure Sockets Layer.

URI Uniform Resource Identifier.

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF