

Střední průmyslová škola elektrotechnická
a Vyšší odborná škola Pardubice

STŘEDNÍ PRŮMYSLOVÁ ŠKOLA ELEKTROTECHNICKÁ

MATURITNÍ PRÁCE – WEBOVÁ APLIKACE

TIME TRACKER

březen 2019

Pavel Junek, 4. C

„Prohlašuji, že jsem maturitní práci vypracoval(a) samostatně a použil(a) jsem literárních pramenů, informací a obrázků, které cituji a uvádím v seznamu použité literatury a zdrojů informací a v seznamu použitých obrázků a neporušil jsem autorská práva.

Souhlasím s umístěním kompletní maturitní práce nebo její části na školní internetové stránky a s použitím jejích ukázek pro výuku.“

V Pardubicích dne

.....

podpis

**Střední průmyslová škola elektrotechnická a Vyšší odborná škola
Pardubice**

MATURITNÍ ZKOUŠKA – PROFILOVÁ ČÁST – MATURITNÍ PROJEKT

zadání maturitní práce

Obor: 18-20-M/01 Informační technologie

Školní rok: 2018/2019

Třída: 4.C

Jméno a příjmení žáka: Pavel Junek

Téma maturitní práce: Time Tracker

Vedoucí maturitní práce: Mgr. Čestmír Bárta

Pracoviště vedoucího: SPŠE a VOŠ Pardubice, Karla IV. 13

Zadání:

Vytvořte webovou aplikaci pro jednoduchou evidenci práce. Aplikace bude zaměstnancům umožňovat zaznamenávat čas odpracovaný na projektech a na základě toho generovat měsíční přehledy. Zaměstnavateli (firmě) poskytne správu klientů, projektů, zaměstnanců, činností a dalších věcí. Také bude umět generovat podklady pro fakturaci klientům na základě evidované práce.

Maturitní práce bude tvořena praktickou částí podle zadání a písemnou prací. Praktická část bude uložena a zpřístupněna na školním serveru.

Součástí praktické části budou:

- responzivní a validní webové stránky umístěné na přidělené adrese na školním serveru včetně ukázkových dat uložených v databázi,
- veškerá administrace databáze proveditelná ve webových stránkách projektu, která bude obsahovat vkládání, editaci i mazání údajů prostřednictvím uživatelských formulářů a bude zahrnovat nahrávání a mazání obrázků či jiných souborů.

Součástí písemné práce bude:

- jméno a příjmení žáka, třída a název práce,
- prohlášení o autorských právech, použitých zdrojích a souhlas s umístěním ukázky webu na školní internetové stránky a souhlas s použitím ukázek pro výuku a souhlas

s poskytnutím projektu za účelem reprezentace školy po dobu následujícího školního roku,

- analýza tématu práce,
- návrh projektu, popis oprávnění přístupu uživatelů, popis použité databáze,
- grafický návrh designu, popis způsobu řešení responzivity webu,
- popis funkcí webových stránek,
- zhodnocení splnění zadání,
- seznam použitých technologií s popisem méně známých technologií, licence k použitému software, skriptům a knihovnám,
- adresa webových stránek umístěných na školním serveru, seznam přístupových údajů registrovaného uživatele a administrátora webu,
- seznam použité literatury, zdrojů informací a zdrojů obrázků použitých na webu,
- seznam obrázků použitých v písemné práci,
- přílohy – E-R diagram databáze, Use Case diagram administrace, screenshoty vytvořených webových stránek.
- pevně vložené a podepsané CD nebo DVD obsahující kopii adresáře webových stránek ze školního serveru, vyexportovanou databázi ve formátu SQL, v případě potřeby popis specifických požadavků na konfiguraci serveru a písemnou práci v elektronické podobě ve formátu PDF a ve formátu DOCX nebo ODT.

Způsob zpracování písemné práce

Zpracování písemné práce bude odpovídat požadavkům dle souboru *Maturitní práce – Formální stránka dokumentace.pdf* dostupným na www.spse.cz. Dodržení stanovených pravidel bude jedním z kritérií hodnocení písemné práce.

Pokyny k obsahu a rozsahu písemné práce

Písemná práce bude obsahovat minimálně 15 stránek vlastního textu (počítáno bez vložených obrázků a ukázek kódu), obrázky větší, než ½ stránky budou uvedeny v příloze. Do tohoto počtu se nezapočítávají úvodní listy, zadání, obsah, seznamy přístupových údajů, technologií, obrázků, literatury a přílohy.

Kritéria hodnocení maturitní práce

Hlavní kritéria

- splnění zadání – práce splňující zadání na méně než 60 % bude hodnocena známkou nedostatečně,
- míra vlastního podílu na řešení,
- náročnost a rozsah práce,
- dodržení stanovených pravidel pro zpracování písemné práce.

Vlastní maturitní práce tvoří jednu část třetí profilové zkoušky; druhou částí je její obhajoba.

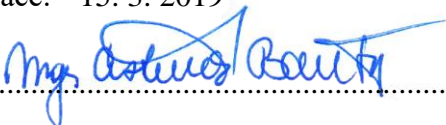
Hodnocení maturitní práce:

- vedoucí maturitní práce a oponent hodnotí maturitní práci podle stanovených kritérií hodnocení známkou výborně až nedostatečně.

Požadavek na počet vyhotovení maturitní práce

Maturitní práci odevzdáte ve stanoveném termínu vedoucímu maturitní práce, a to ve dvou vyhotoveních. Praktickou část ponecháte na školním serveru beze změny.

Termín odevzdání maturitní práce: 15. 3. 2019

Vedoucí maturitní práce: 

Dne: Ing. Ladislav Štěpánek, ředitel školy 

Anotace

Webová aplikace pro jednoduchou evidenci práce, zaměřená převážně na firmy a její zaměstnance. Bude umožňovat správu projektů, klientů. Umožní generovat základní jednoduché podklady pro fakturaci (s možností úpravy počtu hodin).

Klíčová slova: evidence práce, správa firmy, podklady pro fakturaci

Annotation

Web application for simple time tracking, focused on companies and their employees. It will allow project and client management. It will include generation of simple basis for billing (with the possibility to edit hours amount).

Keywords: time tracker, company management, basis for billing

OBSAH

ÚVOD.....	11
1 ANALÝZA OBDOBNÝCH WEBOVÝCH STRÁNEK	12
1.1 TOGGL	12
<i>1.1.1 Kladné stránky.....</i>	<i>12</i>
<i>1.1.2 Záporné stránky.....</i>	<i>13</i>
1.2 TIMEDOCTOR	13
<i>1.2.1 Kladné stránky.....</i>	<i>13</i>
<i>1.2.2 Záporné stránky.....</i>	<i>14</i>
1.3 CLOCKIFY	14
<i>1.3.1 Kladné stránky.....</i>	<i>14</i>
<i>1.3.2 Záporné stránky.....</i>	<i>15</i>
2 NÁVRH PROJEKTU	16
2.1 CÍLOVÉ SKUPINY	16
2.2 POPIS PROJEKTU	16
2.3 USE CASE DIAGRAM	16
2.4 DATABÁZE	17
3 POUŽITÉ TECHNOLOGIE	18
3.1 JAVASCRIPT	18
<i>3.1.1 Node.JS.....</i>	<i>18</i>
<i>3.1.2 TypeScript</i>	<i>18</i>
<i>3.1.3 NPM</i>	<i>19</i>
<i>3.1.4 Express.js</i>	<i>19</i>
<i>3.1.5 Angular.....</i>	<i>19</i>
3.2 MONGODB	20
<i>3.2.1 Mongoose js</i>	<i>20</i>
3.3 PUG.JS	20
3.4 STYLUS	21
4 BACKEND PROJEKTU	22

4.1	ZÁKLADNÍ KONCEPT	22
4.2	STRUKTURA KÓDU.....	22
	4.2.1 Model.....	23
	4.2.2 ValidationSchema.....	23
	4.2.3 Routes.....	24
	4.2.4 Controller	24
4.3	PRINCIP KOMUNIKACE	25
	4.3.1 Komunikace s přihlášeným uživatelem.....	25
5	FRONTEND PROJEKTU	26
5.1	ZÁKLADNÍ KONCEPT	26
5.2	STRUKTURA KÓDU.....	26
	5.2.1 Komponenty.....	26
	5.2.2 Guards (middleware)	28
	5.2.3 Helpers	29
	5.2.4 Models	29
	5.2.5 Pipes.....	30
	5.2.6 Services	30
5.3	LAYOUT.....	31
5.4	ROUTOVÁNÍ.....	31
6	POPIS JEDNOTLIVÝCH ČÁSTÍ APLIKACE	33
6.1	PŘIHLAŠOVACÍ OBRAZOVKA	33
6.2	DASHBOARD (PŘEHLED)	33
6.3	EVIDENCE PRÁCE.....	34
6.4	ADMINISTRÁTORSKÁ ČÁST	35
	6.4.1 Správa uživatelů.....	36
	6.4.2 Správa zákazníků.....	36
	6.4.3 Správa projektů	36
	6.4.4 Správa činností.....	37
	6.4.5 Přehled	37
	6.4.6 Podklady pro fakturaci.....	38
	6.4.7 Vygenerované podklady	39
	6.4.8 Nastavení.....	40

ZÁVĚR	42
SEZNAM PŘÍSTUPOVÝCH ÚDAJŮ	43
SEZNAM OBRÁZKŮ	44
ZDROJE	45
PŘÍLOHY	46

Úvod

Jako maturitní projekt jsem se rozhodl vytvořit jednoduchou aplikaci, která bude mít za cíl umožnit co možná nejrychlejší a nejefektivnější evidenci práce a tím zjednodušit firemní work-flow. Také bude umět generovat základní podklady pro fakturaci kvůli zrychlení vydávání faktur klientům firmy.

Vím, že podobných aplikací se dá na internetu najít spousta. Ale žádná z nich nepokryje jednoduchou správu všech bodů zmíněných v zadání. A pokud ano, tak buď velmi složitě, nebo se jedná o poměrně nákladné řešení. Z toho důvodu jsem si vybral právě toto zadání.

Já sám pracuji (nebo jsem pracoval) s několika nástroji pro evidenci práce. Žádný z nich ale není ideální. Některé jsou zaměřené převážně na firmy (a manažery), takže jsou pro běžné pracovníky zbytečně složité. A jiné jsou zase určené jednotlivcům, které zase nenabízí mnoho prostoru pro firmy (a například pro projekt manažery).

Rozhodl jsem se tedy vytvořit vlastní jednoduchou aplikaci pro evidenci práce. Pokud projekt uspěje, navrhnu jeho použití i firmě, se kterou nyní dlouhodobě spolupracuji a budu se snažit projekt udržovat a vylepšovat i do budoucna. (Jako budoucí vylepšení se nabízí například mobilní nebo desktopová nativní aplikace.)

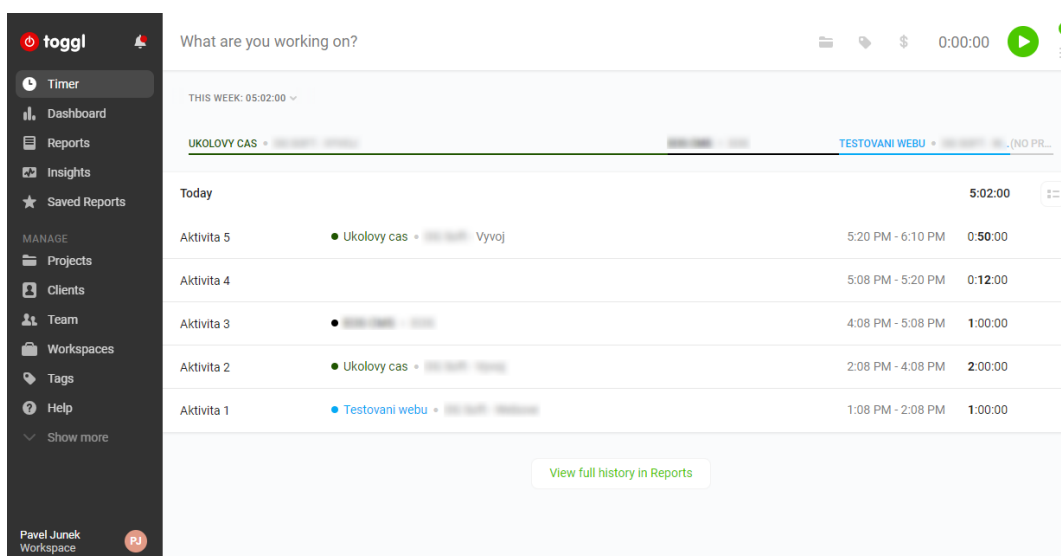
Celá aplikace bude napsaná v jazyce JavaScript. Z toho důvodu, že v něm vidím asi největší budoucnost, a proto bych se v něm rád zdokonalil a ukázal, že lze poměrně jednoduše vytvořit celou aplikaci (backend i frontend) v jednom programovacím jazyce.

1 Analýza obdobných webových stránek

Analýzu je důležité provádět při každém vývoji nové aplikace zejména proto, abychom zjistili, jakou můžeme nabídnout konkurenční výhodu, čemu se vyhnout, anebo co je dobré z ostatních projektů převzít, případně vylepšit.

1.1 Toggl

Adresa: toggl.com



Obrázek 1: Toggle (zdroj: vlastní)

Současně asi nejlepší aplikace pro evidenci práce, kterou sám běžně používám. Umožňuje členění do projektů, správu klientů nebo týmů. Dle mého je ale zaměřená více na jednotlivce (živnostníky), než na firmy. Vhodná je ale i pro menší týmy. Jde také pravděpodobně o nejpoužívanější aplikaci pro vykazování práce mezi freelancery a drobnými podnikateli.

1.1.1 Kladné stránky

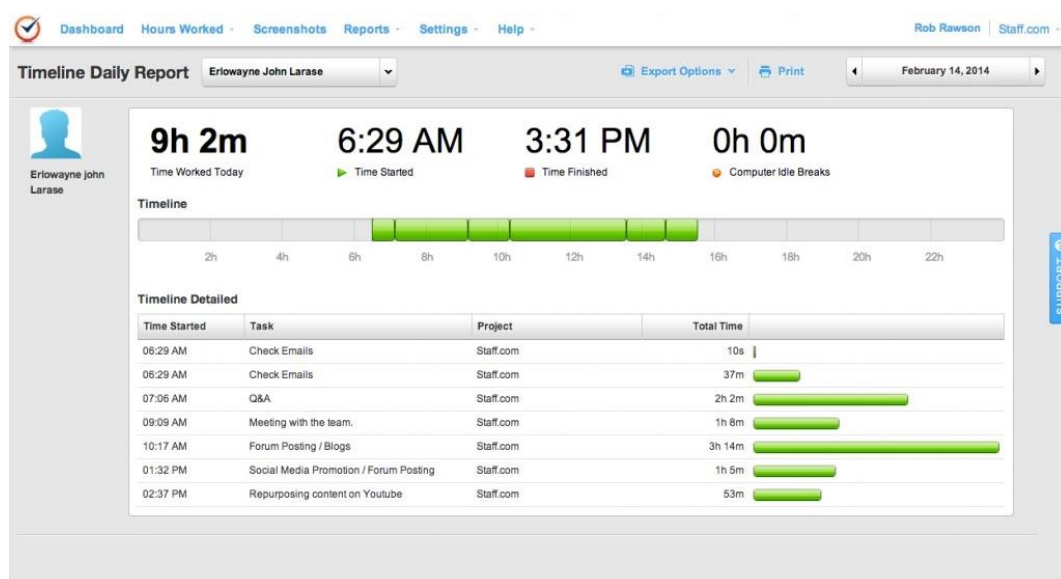
- Podrobné generování přehledů
- Velmi jednoduchá evidence a práce s daty
- Multiplatformní

1.1.2 Záporné stránky

- Většina funkcí dostupných pouze v placené verzi (členové týmu, money management apod.)
- Na vygenerovaném přehledu nelze editovat částky nebo počet hodin

1.2 TimeDoctor

Adresa: timedoctor.com



Obrázek 2: TimeDoctor (zdroj: alternativeto.net)

Aplikace zaměřená spíše na společnosti než jednotlivce, která nabízí kromě evidence práce také například kompletní monitoring zaměstnanců (včetně monitorování myši s klávesnicí, navštívených webových stránek nebo sledování polohy přes GPS). Tím by měla zařídit větší produktivitu zaměstnanců. Dle mého názoru jde ale spíš o nástroj pro snížení pracovního nadšení a zhoršení příjemných pracovních podmínek. Osobně bych asi nechtěl pracovat pro firmu, která by využívala tento nástroj.

1.2.1 Kladné stránky

- Integrace se spoustou ostatních služeb (např. Google Apps, Basecamp, Jira, Trello a další)
- Obsahuje jednoduchou správu úkolů

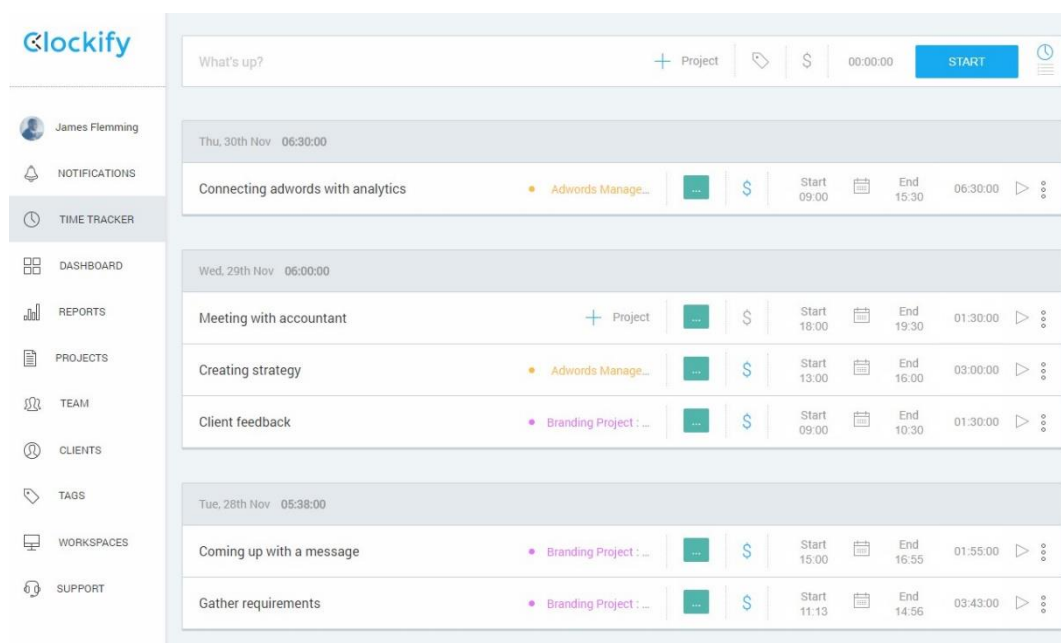
- Zaměstnavatelé mají kompletní přehled o svých podřízených
- Vlastní API

1.2.2 Záporné stránky

- Kompletní monitoring může být pro zaměstnance velmi nepříjemný
- Nenabízí verzi zdarma
- Starší design

1.3 Clockify

Adresa: clockify.me



Obrázek 3: Clockify (zdroj: alternativeto.net)

Poměrně nová jednoduchá webová aplikace velmi podobná aplikaci Toggl. Umožňuje základní timetracking, dělení do týmů a projektů, umí generovat reporty (v podstatě stejné jako Toggl). Oproti dříve zmíněným aplikacím je ale zcela zdarma. Mohou jí používat jak jednotlivci, tak menší firmy.

1.3.1 Kladné stránky

- Zcela zdarma

- Umožňuje ruční zadávání odpracovaných hodin do týdenních timesheetů
- Minimalistický a čistý design
- Integrace s ostatními službami (Asana, Trello, Gmail, ...)

1.3.2 Záporné stránky

- Uživatelé si občas stěžují na rychlost

2 Návrh projektu

2.1 Cílové skupiny

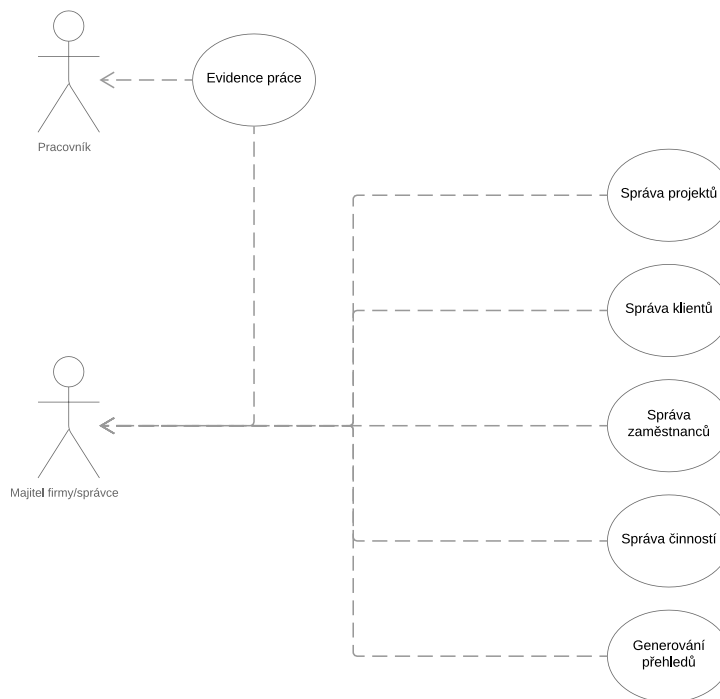
Firmy a společnosti, které chtějí jednoduchý time tracker pro své zaměstnance.

2.2 Popis projektu

Projekt bude rozdělen na dva samostatné celky. Backend (server) a frontend (klient). Backend bude poskytovat data libovolnému počtu klientů pomocí API. Bude komunikovat s databází a řešit veškerou aplikační logiku. Frontend bude pouze „grafickým nástrojem“ zprostředkujícím komunikaci mezi uživatelem a serverovou API. V rámci maturitního projektu bude vytvořen jeden klient – a to webová stránka.

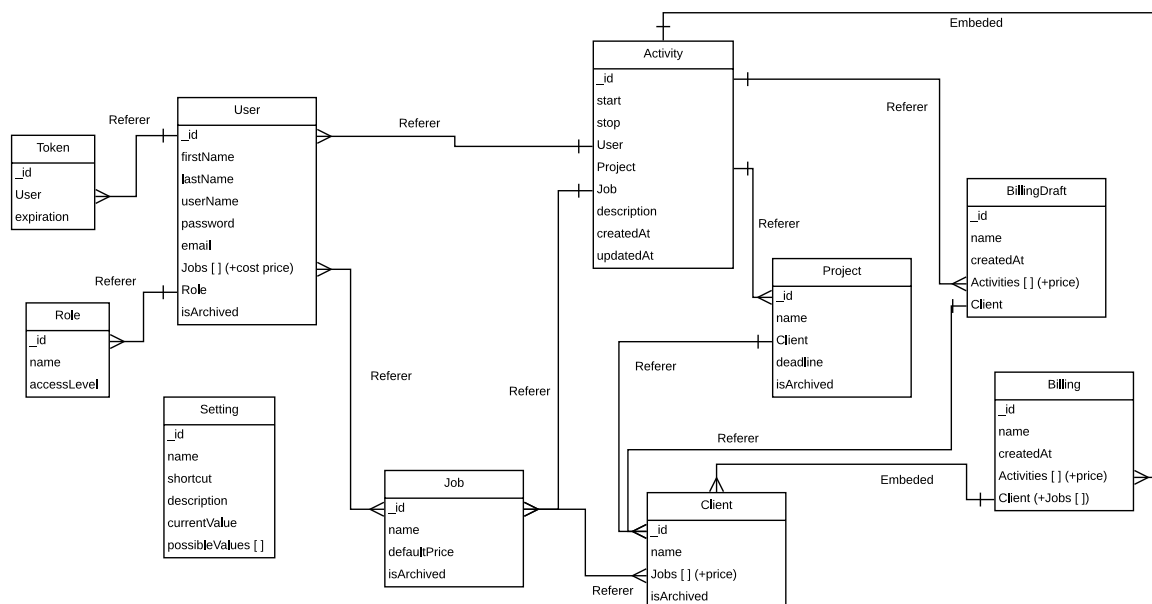
2.3 Use Case diagram

Time Tracker - Use case diagram



Obrázek 4: Use Case diagram

2.4 Databáze



Obrázek 5: struktura databáze

Na výše uvedeném diagramu databáze jsou vidět všechny potřebné dokumenty (tak se v NoSQL říká něčemu, jako jsou tabulky v relačních databázích). Jelikož NoSQL nemá žádnou standardizovanou formu pro modelování grafů, snažil jsem se napodobit ERD relačních databází. U některých částí to byl docela problém, tak se pokusím nastínit, jak je diagram zamýšlen.

Jednotlivé dokumenty jsou propojeny „relacemi“, u každé je uveden typ vazby. Pro přehlednost bude v příloze také uveden dump databáze, který bude dle mého názoru v tomto případě mnohem více vypovídající než diagram.

3 Použité technologie

3.1 JavaScript

Celý projekt je napsaný v programovacím jazyce JavaScript, respektive v jazycích z něj vycházejících nebo jeho frameworků.

JavaScript původně sloužil k doplnění značkovacího jazyka HTML (jednoduché efekty, manipulace s DOM apod.). Nyní jde o plnohodnotný samostatný jazyk, který lze využít jak na klientovi, tak na serveru. Důležité je neplést si ho s Javou, se kterou nemá vůbec nic společného (tedy kromě podobné syntaxe).

Čistý JavaScript v projektu použiji jenom minimálně. Budu používat jeho serverovou mutaci, Node.JS, a nadstavbu TypeScript.

3.1.1 Node.JS

Jde o JavaScript, který neběží na klientovi (jak bývá zvykem), ale na serveru. V současnosti je velmi rozšířený a používáný společnostmi, jako jsou například Netflix, LinkedIn, PayPal, eBay nebo Trello.

Veliký rozdíl oproti většině programovacích jazyků je jeho asynchronnost. Jednoduše řečeno to znamená, že se „všechno děje najednou“. Oproti jazyku PHP, kde se kód vykonává v takovém pořadí, v jakém byl napsán. Další řádek kódu se nezačne zpracovávat, dokud se nedokončil předchozí řádek. Tohle v Node.JS neplatí. Může se zpracovávat klidně padesát řádků najednou (a na dokončení operace se reaguje pomocí Promises).

Jelikož Node.JS běží na serveru, není potřeba řešit podporu starších prohlížečů a je možné psát kód v JavaScriptu podle nejnovějších standardů (v současnosti ES6) bez následné kompilace zdrojových kódů (např. pomocí nástroje Babel.js).

3.1.2 TypeScript

Jde o nadstavbu JavaScriptu od společnosti Microsoft, který se kompiluje do JavaScriptu. Přináší podporu pro klasické programátorské konstrukce, které klasický

JavaScript neumožňuje. Například statické typování proměnných, dekorátory, třídy, interface, moduly, a další.

Použit je jak při tvorbě serverové části, tak i při tvorbě části klientské.

3.1.3 NPM

Jedná se o nástroj pro správu balíků v JavaScriptu. V podstatě jde o obdobu Composeru pro PHP. Umožňuje jednoduše instalovat a spravovat balíky a závislosti aplikací přes příkazový řádek.

3.1.4 Express.js

Express je asi nejpoužívanější serverový framework pro Node.JS. Je určený převážně k tvorbě backendu a API. I přes to, že je velmi minimalistický, umí toho poměrně dost. Poskytuje například podporu pro middleware, nebo jednoduchou správu http requestů.

Je na něm postaveno mnoho dalších frameworků a dalo by se říct, že jde o standard při tvorbě serveru v Node.JS. Ve svém projektu ho při tvorbě serveru použijí tedy samozřejmě také.

3.1.5 Angular

JavaScriptový framework pro tvorbu SPA (Single Page Application – celá aplikace je obsažena v jednom HTML a jednom JS souboru) frontendových aplikací. Autorem je společnost Google, která ho také hojně využívá ve svých projektech – např. Google Pay, YouTube, Google Cloud, AdWords, Framework je také velmi využíván například Microsoftem.

Využívá populární návrhový vzor MVC (model-view-controller – odděluje aplikační logiku od zobrazených šablon). Díky tomu, že funguje jako SPA, jsou weby v něm napsané velmi rychlé.

Kromě jiného vývojářům umožňuje data-binding, poskytuje vlastní rozšíření do příkazového řádku, má podporu pro testování kódu,

Ve výchozím stavu Angular používá TypeScript, HTML a CSS. V projektu je TypeScript zachován, místo klasického HTML a CSS jsou použity preprocesory PugJS a Stylus.

3.2 MongoDB

Místo klasické relační databáze je použita databáze typu NoSQL. Konkrétně MongoDB. Oproti relační databázi data neukládá do tabulek, ale do souborů (dokumentů) ve formátu podobném JSONu (u MongoDB to je BSON).

Rozdílů je ale samozřejmě mnohem více. MySQL má pevně danou strukturu (tabulky, sloupce, datové typy, ...), NoSQL nic takového nemá. Musí to vyřešit až vývojář na úrovni aplikace. V NoSQL existují (oproti MySQL) dva typy vazeb – embed (vložení) a reference (odkaz). Zatímco reference je obdoba vazby přes cizí klíče v MySQL (ovšem bez hlídání integrity dat – opět je potřeba vyřešit na straně aplikace), vazba embed znamená, že „entita“ obsahuje „druhou entitu“ přímo v sobě.

Samozřejmě, že rozdílů je ještě mnohem více. Veliká výhoda MongoDB (nebo jakékoliv jiné NoSQL databáze) je rychlost. Používá se proto například při práci s „big data“. MongoDB používají například společnosti eBay, Adobe, Google, Cisco a spousta dalších.

3.2.1 Mongoose js

Knihovna pro modelování MongoDB objektů v NodeJS. Kromě jiného umožňuje správu datových typů a zajišťuje celou komunikaci mezi NodeJS a MongoDB. Včetně vyhledávání dat, vkládání, mazání, tvorby dokumentů nebo indexů a dalších akcí.

3.3 Pug.js

Pug.js (dříve Jade) je preprocesor pro HTML a velmi používaný nástroj pro tvorbu statických HTML šablon. Jedná se o indent-based jazyk. To znamená, že odstraňuje závorky (< a >) a ukončovací tagy. Místo toho se vše zapisuje pomocí odsazení řádků tabulátorem. Kromě jiného integruje například cykly, nebo přímý zápis JS. Kompiluje se do HTML. Kód vypadá například následovně:

```
doctype html
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) bar(1 + 5)
  body
    h1 Pug - node template engine
    #container.col
      if youAreUsingPug
        p You are amazing
      else
        p Get on it!
    p.
      Pug is a terse and simple templating language with a
      strong focus on performance and powerful features.
```

(zdroj: www.npmjs.com)

3.4 Stylus

Preprocesor pro CSS styly. Podporuje proměnné, mixiny (obdoba funkcí z programovacích jazyků), nesting (zanořování elementů), podmínky, cykly a mnoho dalších.

Oproti klasickému CSS má velmi zjednodušenou syntaxi. Není potřeba psát složené závorky, ani středníky na konci řádků. Je možné dokonce vynechat i dvojtečky. Kód pak vypadá například takto:

```
border-radius(n)
  -webkit-border-radius n
  -moz-border-radius n
  border-radius n

body
  h1
    border 1px solid red
    border-radius 5px
```

(zdroj: www.stylus-lang.com)

4 Backend projektu

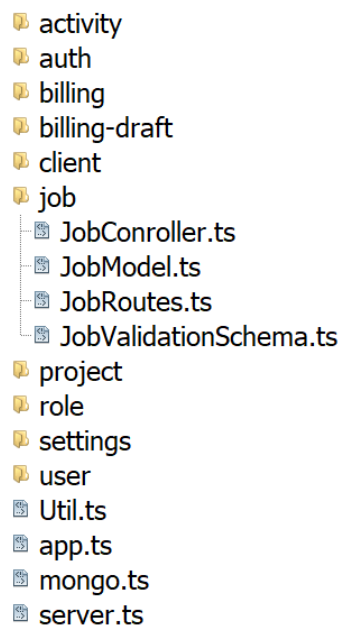
4.1 Základní koncept

Backend projektu je napsaný formou API (Application Programming Interface) v programovacím jazyce TypeScript s využitím frameworku Express.js a NoSQL databázi MongoDB pro uchovávání dat. O kvalitu a integritu kódu se stará TSLint (linter pro TypeScript).

4.2 Struktura kódu

Aplikace je rozdělená do modulů, kde každý modul v podstatě odpovídá jednomu dokumentu v databázi (obdoba MySQL tabulky). Moduly jsou postavené na zjednodušené MVC architektuře (s absencí view). Každý modul obsahuje minimálně čtyři části:

- Model – definuje schéma MongoDB dokumentu (obdoba entity při použití MySQL)
- ValidationSchema – definuje a validuje uživatelské vstupy a URL parametry
- Routes – definuje dostupné API endpointy (většinou na bázi CRUD)
- Controller – obsluhuje API endpointy



Obrázek 6: Struktura backendu

4.2.1 Model

Model v podstatě odpovídá Mongoose schématu a určuje strukturu NoSQL dokumentu. Definuje atributy (obdoba sloupců v MySQL) a jejich typy, indexy nebo validační pravidla. Může obsahovat takzvané „hooky“ – funkce (callbacky), které se vyvolají při určité události (při validaci schématu, před uložením do databáze apod.). Kromě toho může mít i klasické metody (podle MVC architektury).

```
const Schema = mongoose.Schema;

const JobSchema = new Schema({
  name: {
    type: String,
    required: "Enter a name",
    unique: true,
  },
  defaultPrice: {
    type: Number,
    required: "Enter default price",
  },
  isArchived: {
    type: Boolean,
    required: true,
    default: false,
  },
});

export const Job = mongoose.model("Job", JobSchema);
```

(ukázka kódu modelu)

4.2.2 ValidationSchema

Obstarává validaci requestů na API endpointy za využití externí knihovny Joi. Kromě Joi je použita také jednoduchá knihovna Celebrate, aby validační schémata mohla být dále použita jako middleware.

Pokud request odpovídá validačním pravidlům, middleware jej propustí dále. Pokud ne, pošle okamžitě odpověď (response) s chybovým kódem a zprávou o špatně zadaných údajích a zabrání dalšímu zpracování.

```
export const JobValidationSchema = celebrate({
  body: Joi.object().options({ abortEarly: false }).keys({
    name: Joi.string(),
    defaultPrice: Joi.number(),
    isArchived: Joi.boolean(),
  }),
});
```

(ukázka kódu validačního schéma)

4.2.3 Routes

S využitím frameworku Express definuje API endpointy a pomocí middleware řídí přístupy k nim. Například kontroluje, zda je uživatel přihlášen, jestli má potřebná práva, nebo integruje výše zmíněné validační schémata. Jedna ruta odpovídá právě jedné funkci/metodě v kontroleru (viz dále).

```
export class JobRoutes {

  public jobController: JobController = new JobController();

  public routes(app: express.Application): void {
    app.route("/job").post(
      AuthService.jwtAuth(),
      AuthService.accessLevel(RoleAdmin),
      JobValidationSchema,
      this.jobController.createJob,
    );

    app.route("/job/:jobId").get(
      AuthService.jwtAuth(),
      JobIdValidationSchema,
      this.jobController.getJob,
    );
  }
}
```

(ukázka kódu třídy s routami)

4.2.4 Controller

Třída, která obstarává obsluhu API endpointů a řeší komunikaci s databází. Má na starosti také odpovědi serveru na requesty.


```

public createJob(req: Request, res: Response) {
  Job.init()
    .then(() => Job.create(req.body))
    .then(job => res.status(201).send())
    .catch(e => {
      if (e.name === "MongoError" && e.code === 11000)
        res.status(400).send({message: "Činnost už existuje"});
      res.status(400).send(e);
    });
}

```

(ukázka kódu metody obsluhující vytvoření nového záznamu v databázi)

4.3 Princip komunikace

Veškerá data, která API přijímá nebo poskytuje jsou ve formátu JSON (JavaScript Object Notation), který je v podstatě spolu s XML standardem pro jakoukoliv API komunikaci.

Komunikace a ověřování uživatele probíhá na bázi JWT tokenů. K autorizaci uživatele je integrována knihovna Passport.js.

4.3.1 Komunikace s přihlášeným uživatelem

Uživatel se přihlašuje klasicky jménem a heslem. Tyto údaje se ověří vůči databázi, a pokud přihlášení proběhne úspěšně, vygeneruje se token a uloží se do databáze. Poté se používá pro ověření uživatele při další komunikaci s API. Platnost tokenu je časově omezena a obnovuje se při každém použití. Pokud se uživatel odhlásí, token se odstraní z databáze a už není možné jej znovu použít.

5 Frontend projektu

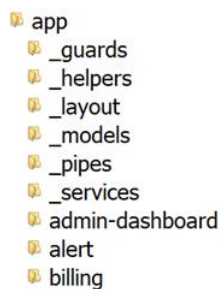
5.1 Základní koncept

Webová aplikace je postavena na JavaScriptovém frameworku Angular, pro tvorbu HTML šablon používá Pug.js a pro stylování Stylus. Angular doplňuje designový framework Clarity Design System od společnosti VMware. O kvalitu a integritu kódu se stará opět TSLint.

5.2 Struktura kódu

Angular pracuje s komponentami, přičemž platí, že komponenty by měly být samostatně použitelné, co možná nejmenší jednotky, ze kterých se poté skládají stránky (také komponenty). Kromě komponent aplikace používá i jiné specifické typy tříd:

- Guards: to samé jako middleware, řídí přístupy při routování
- Helpers: často používané užitečné funkce/třídy, nebo interceptory (viz dále)
- Models: modely podle MVC architektury
- Pipes: definice filtrů dostupných pro použití v šablonách
- Services: služby, které zajišťují např. komunikaci s API



```
app
├── _guards
├── _helpers
├── _layout
├── _models
├── _pipes
├── _services
├── admin-dashboard
├── alert
└── billing
```

Obrázek 7: Struktura kódu frontentu

5.2.1 Komponenty

Angular odděluje aplikační část od zobrazovací. Komponenta je TypeScriptová třída, která odpovídá kontroleru podle architektury MVC a načítá HTML šablonu (v tomto případě zkompilovanou z Pugu) a volitelně i soubor se styly (Angular má implementovanou podporu

několika CSS preprocesorů, v aplikaci je použit Stylus). Každá komponenta tedy může mít vlastní styly, které nijak neovlivní zbytek aplikace.

5.2.1.1 Komponenta jako kontroler

Třída komponenty je dekorovaná dekorátorem `Component`, kde se povinně definuje selektor pro použití v jiných komponentách, volitelně například použitá HTML šablona (nebo přímo HTML kód komponenty), použité styly apod. Komponenta může mít libovolný počet vstupních a výstupních parametrů, definovaných pomocí dekorátorů.

Angular používá vlastní DI (dependency injection) kontejner, proto mohou být do komponenty přes konstruktor injektovány různé závislosti (například služby).

```
@Component({
  selector: 'app-project-filters',
  templateUrl: './project-filters.component.html',
})
export class ProjectFiltersComponent implements OnInit {

  @Input() activeFilters: ProjectFilters;

  public availableClients: Client[];

  constructor(private clientService: ClientService) { }

  ngOnInit(): void {
    this.loadAvailableClients();
  }

  private loadAvailableClients() {
    this.clientService.getAll(false)
      .subscribe(clients => {
        this.availableClients = clients;
      });
  }
  ...
}
```

(ukázka části kódu komponenty – zkráceno)

5.2.1.2 Šablona komponenty

Šablony jsou psané pomocí preprocesoru Pug a následně se kompilují do HTML, které načítá třída komponenty. Šablona má přístup k veřejným vlastnostem a metodám komponenty. Angular podporuje pokročilý data-binding – například pokud se změní hodnota proměnné v komponentě, změna se automaticky v reálném čase projeví i v šabloně.

```
clr-modal('[(clrModalOpen)]'="alertsService.message")
  h3.modal-title {{alertsService.message}}
```

(ukázka kódu šablony v Pugu)

```
<clr-modal [(clrModalOpen)]="alertsService.message">
  <h3 class="modal-title">{{alertsService.message}}</h3>
</clr-modal>
```

(ukázka vygenerovaného neminifikovaného HTML)

5.2.1.3 Styly komponenty

Pokud komponenta potřebuje jiné styly než výchozí, může použít své vlastní specifické styly, které jsou platné jenom pro tutéž komponentu. Případně Angular definuje speciální CSS selektor, který umožňuje stylovat i obsažené komponenty (potomky).

```
:host /deep/
  .datagrid-filter-close-wrapper
    display none
  clr-select-container
    margin-top 0

@media all and (max-width 768px)
  .nav-link.nav-icon
    opacity 1

  clr-icon
    color #006a91
    width 2.5rem
    height 2.5rem
```

(ukázka stylů komponenty)

5.2.2 Guards (middleware)

V názvosloví Angularu neexistuje middleware, pouze guard. Funkcionálně jde ale o naprosto totožné části. Na základě zadaných parametrů řídí, zda má uživatel právo na požadovanou akci (např. zobrazení stránky).

```

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {

  constructor(private router: Router) { }

  canActivate(next: ARS, state: RSS): boolean {
    if (localStorage.getItem('user'))
      return true;

    this.router.navigate(['/login'], {queryParams: {returnUrl: state.url}});
    return false;
  }
}

```

(ukázka kódu guardu pro kontrolu přihlášení uživatel – zkráceno a upraveno)

5.2.3 Helpers

Třídy, funkce, nebo konstanty, které svou podstatou nezapadají jinam do logického rozdělení aplikace. Kromě jiného tato část aplikace obsahuje i tzv. interceptory. Jsou to třídy, které zachytávají http requesty a modifikují je (před odesláním, nebo po přijetí odpovědi). Typickým příkladem je přidávání ověřovacího tokenu uživatele do hlavičky každého requestu, nebo odchytávání chybových stavů v odpovědi.

```

@Injectable()
export class JwtInterceptor implements HttpInterceptor {

  constructor(private authService: AuthenticationService) { }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable {
    const user = this.authService.getCurrentUser();
    if (user && user.token) {
      req = req.clone({
        headers: {
          Authorization: user.token
        }
      });
    }
    return next.handle(req);
  }
}

```

(ukázka kódu interceptoru, který přidává token do hlavičky – zkráceno)

5.2.4 Models

V podstatě obyčejné TypeScriptové třídy odpovídající modelům podle MVC architektury. Ve většině případů model obsahuje stejné vlastnosti, jako data přijatá z API.

```
export class User {
  _id: string;
  userName: string;
  firstName: string;
  lastName: string;
  email: string|null;
  roleId: string;
  jobs: UserJob[];
  isArchived: boolean;

  public getFullName(): string {
    return `${this.firstName} ${this.lastName}`;
  }
}
```

(ukázka kódu modelu uživatele – zkráceno)

5.2.5 Pipes

Jde o specifické třídy, které definují filtry, jenž umožňují snadnou modifikaci dat v šablonách.

```
@Pipe({
  name: 'msAsString'
})
export class MsAsStringPipe implements PipeTransform {

  transform(ms: number): string {
    const
      minutes = Math.trunc((ms / (1000 * 60)) % 60),
      hours = Math.trunc((ms / (1000 * 60 * 60)) % 24);

    return
    `${(hours<10)?'0'+hours:hours}:${(minutes<10)?'0'+minutes:minutes}`;
  }
}
```

(ukázka pipe pro převod času v milisekundách na lidsky čitelný formát – upraveno)

```
.card
  .card-header Celkem
  .card-block
    .card-title {{totalTime | msAsString}} hodin
```

(ukázka části kódu šablony s použitím pipe)

5.2.6 Services

Služby, které zajišťují veškerou komunikaci v aplikaci. Ať už jde o komunikaci mezi jednotlivými komponentami, nebo o napojení na API. Každá služba je dekorována

dekorátorem Injectable, díky čemuž se přidá do závislostí v DI kontejneru a může být dále předávána (injektována) jiným třídám jako závislost.

```
@Injectable({
  providedIn: 'root'
})
export class SettingsService {

  constructor(private httpClient: HttpClient) { }

  getAll() {
    return this.httpClient.get<Setting[]>(`${env.apiUrl}setting`);
  }
  ...
}
```

(ukázka části služby pro komunikaci s API – zkráceno a upraveno)

5.3 Layout

Jelikož aplikace nepoužívá jenom jeden typ layoutu (rozložení), jsou všechny layouty definovány jako samostatné komponenty s rozdílným HTML a stylováním. O použití layoutu rozhoduje router (viz dále). Aplikace používá celkem dva (respektive tři) různé typy rozložení:

- Výchozí – použito na dashboardu
- Admin – použito pro všechny stránky v administraci
- Bez layoutu – komponenta používá svůj vlastní specifický layout (příkladem je přihlašovací stránka)

5.4 Routování

Veškeré směrování, řízení přístupů, hezké URL adresy a použití layoutů obstarává speciální modul. Každá ruta může mimo jiné obsahovat následující parametry:

- path: URL adresa stránky
- component: třída komponenty, která se použije pro zobrazení stránky
- canActivate: guards pro řízení přístupu
- children: potomci (routy)

```

{
  path: 'admin',
  component: AdminLayoutComponent,
  canActivate: [AuthGuard, AdminGuard],
  children: [
    { path: '', component: AdminDashboardComponent, pathMatch: 'full' },
    { path: 'jobs', component: JobsComponent },
    { path: 'projects', component: ProjectsComponent },
    { path: 'clients', component: ClientsComponent },
    { path: 'users', component: UsersComponent },
    { path: 'overview', component: OverviewComponent },
    { path: 'billing-drafts', component: BillingDraftComponent },
    { path: 'billing', component: BillingComponent },
    { path: 'settings', component: SettingsComponent },
  ],
},

```

(ukázka části definice rout)

Na výše uvedené ukázce kódu lze vidět, jak vypadají routy pro administraci. Díky použití zanořování rout je možné definovat layout i guardy pro všechny komponenty najednou. Příklad pro komponentu *JobsComponent*:

- bude dostupná z URL adresy */admin/jobs*
- bude dostupná pouze pro přihlášeného uživatele s právy administrátora (definováno parametrem (polem) *canActivate*)
- nejprve se načte komponenta *AdminLayoutComponent*, do které se následně vloží obsah komponenty *JobsComponent*

6 Popis jednotlivých částí aplikace

6.1 Přihlašovací obrazovka

Přihlašovací obrazovka je minimalistická, bez zbytečných elementů. Jelikož aplikace slouží výhradně pro firemní účely, jde o jedinou stránku dostupnou bez přihlášení.

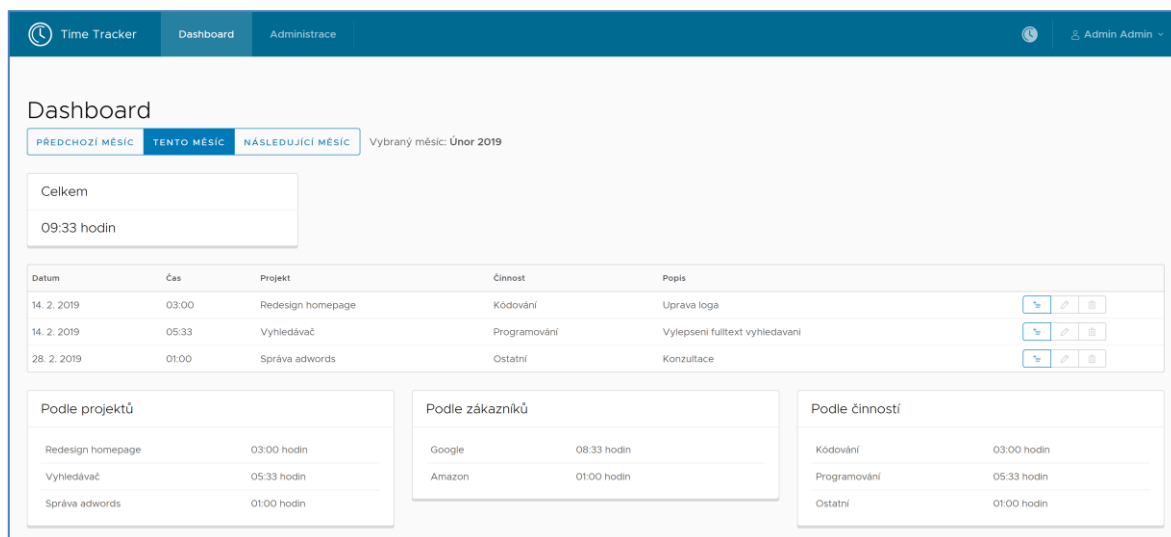
Kromě přihlašovacího formuláře neobsahuje vůbec nic. Registrace není potřeba – dokonce je v tomto případě nežádoucí. Nové uživatele může vytvářet pouze přihlášený administrátor.



Obrázek 8: Přihlašovací obrazovka

6.2 Dashboard (přehled)

Po přihlášení uvidí uživatel tuto stránku. Obsahuje rychlý přehled jeho evidovaných aktivit paginovaný podle měsíců. V prostřední části obrazovky se nachází tabulka se všemi zaevidovanými aktivitami. U každé aktivity může uživatel zvolit „rychlou evidenci“ – otevře se dialog s evidencí s předvyplněnými hodnotami dle dané aktivity. Pokud byla aktivita zaevidována před méně, než jedním dnem (nebo ještě nebyla vyfakturována), může ji uživatel upravit, nebo odstranit.



Obrázek 9: Dashboard

6.3 Evidence práce

Zaznamenávání aktivit probíhá pomocí dialogového okna, které se otevírá kliknutím na ikonu hodin v hlavičce. Uživatel vyplní datum a čas, poté vybere ze seznamu zákazníka. Na základě toho se načtou dostupné projekty. Dále musí uživatel vybrat projekt, činnost a vyplnit popis.

Pokud uživatel okno zavře (nebo potvrdí evidenci), údaje, které vyplnil, zůstanou zapamatovány (i při procházení napříč aplikací) a při znovu otevření dialogu se předvyplní. Je tak velmi jednoduché zaznamenávat práci např. na stejném projektu i ve větším množství.

Evidence nové práce (ručně)

Čas

01:00

Datum

10. 3. 2019

Zákazník

Google

Projekt

Vyhledávač

Činnost

Programování

Popis

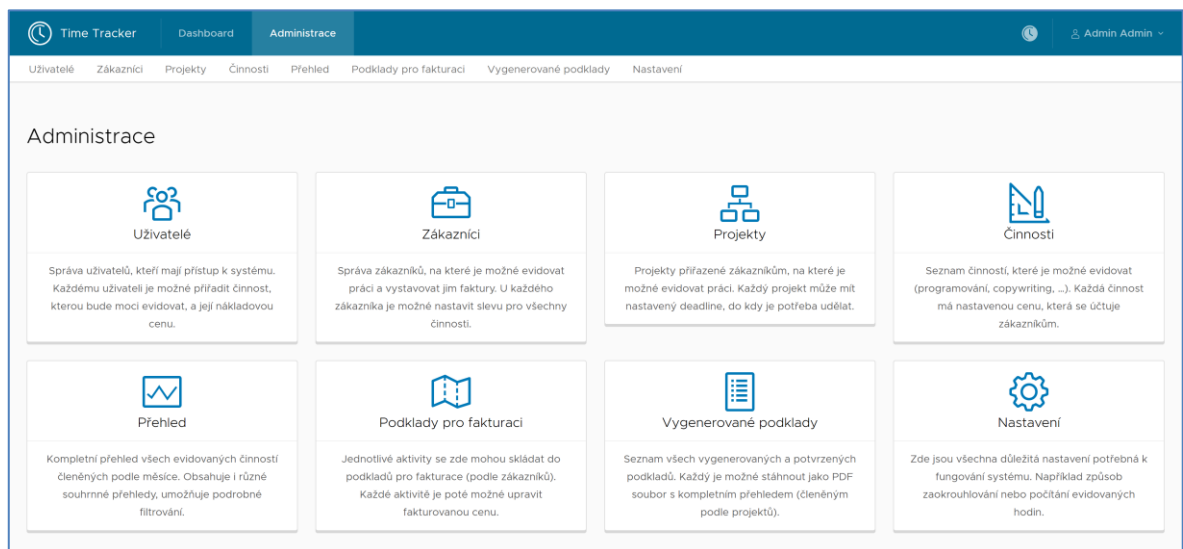
Popis činnosti

OK

Obrázek 10: Dialog pro evidenci práce

6.4 Administrátorská část

Zbytek webu je dostupný administrátory. Po přepnutí se do části administrace uvidí uživatel rozcestník s dostupnými „moduly“, u každého s popisem toho, k čemu je určen.

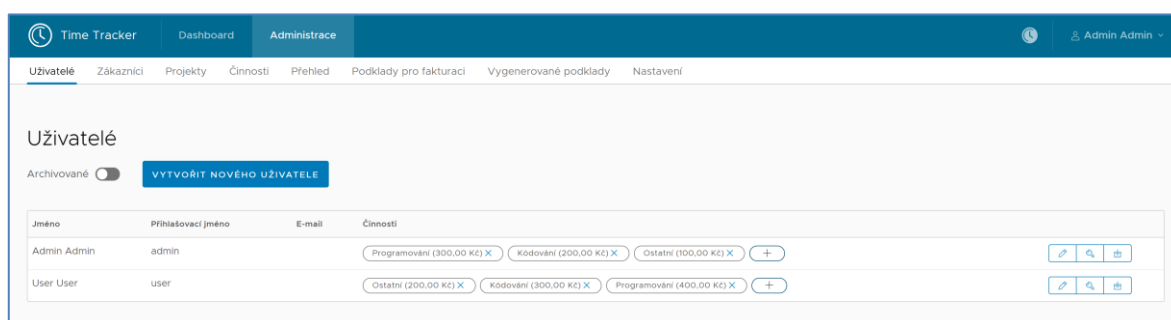


Obrázek 11: Administrační rozcestník

6.4.1 Správa uživatelů

Jednoduchý modul umožňující spravovat uživatele. Každému uživateli je kromě jiného možné přiřadit činnosti (s nákladovou cenou), na které bude poté moci evidovat práci. Zakládají se zde i nové uživatelské účty.

Existujícího uživatele není možné smazat – pouze archivovat. Archivovaní uživatelé se zobrazí přepnutím filtru nad tabulkou.



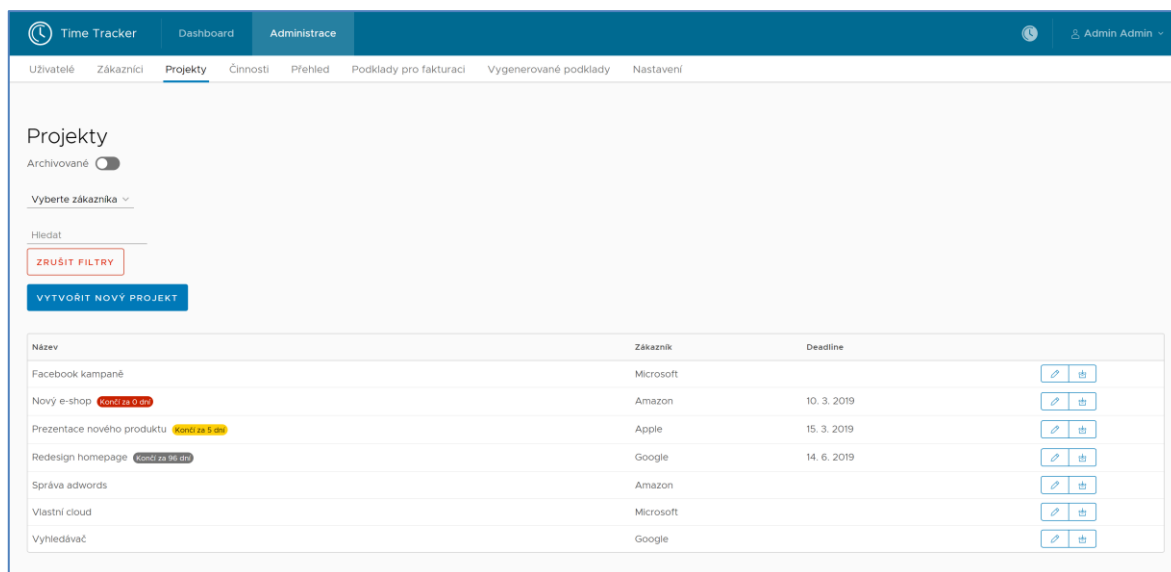
Obrázek 12: Správa uživatelů

6.4.2 Správa zákazníků

Modul velmi podobný správci uživatelů. Na úplně stejném principu se zde přiřazují činnosti zákazníkům, s možnou přenastavenou fakturační cenou. Místo mazání je opět dostupná pouze archivace.

6.4.3 Správa projektů

Opět velmi podobný modul předchozím dvěma. Obsahuje rozšířenější filtry – je možné vyhledávat podle názvu projektů a filtrovat podle zákazníka. Každý projekt může mít nastavený deadline – takový projekt je označen barevnou indikací s počtem zbývajících dnů do dokončení.



Obrázek 13: Správa projektů

6.4.4 Správa činností

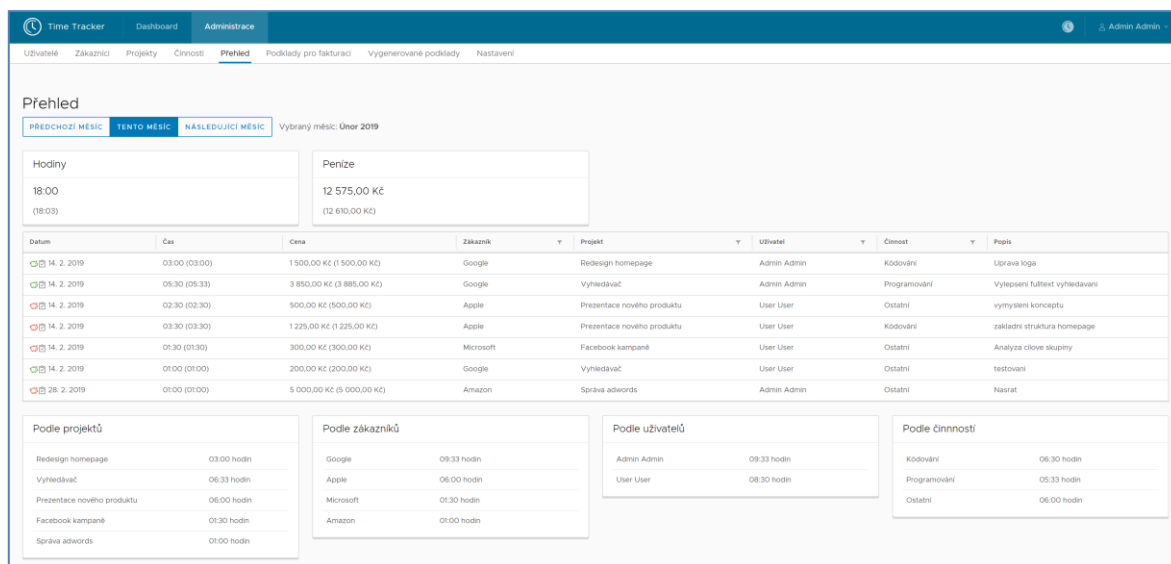
Velmi jednoduchá správa činností na stejném principu jako předešlé moduly. Opět je dostupná pouze archivace.

6.4.5 Přehled

Jde v podstatě o rozšířený modul dashboard. Obsahuje kompletní přehled všech zaevidovaných aktivit od všech uživatelů. Mezi aktivitami je možné filtrovat podle zákazníku, projektu, uživatele nebo činnosti.

Čas a cena jsou u každé aktivity zaokrouhlené podle pravidel z nastavení (viz dále), ale zobrazují se i originální nezaokrouhlené hodnoty. U každé aktivity se nachází indikátor, který určuje, jestli už byla aktivita vyfakturována – ikona pokladničky (prasátka) svítí zeleně (vyfakturováno), nebo červeně.

Pod tabulkou se seznamem aktivit se nachází souhrnný přehled podle různých kritérií (podle projektů, zákazníků, uživatelů a činností)



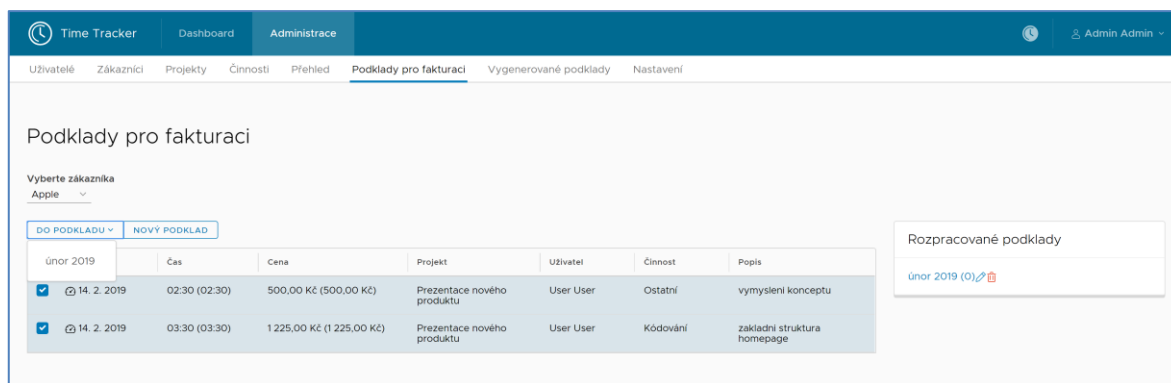
Obrázek 14: Přehled aktivit

6.4.6 Podklady pro fakturaci

Jedná se o asi nejsložitější modul v celé aplikaci. Z nevyfakturovaných aktivit se zde skládají podklady pro fakturaci.

Po otevření modulu musí uživatel nejprve zvolit zákazníka. Následně se mu zobrazí stránka s přehledem nevyfakturovaných aktivit a rozpracovaných podkladů (náležící vybranému zákazníkovi).

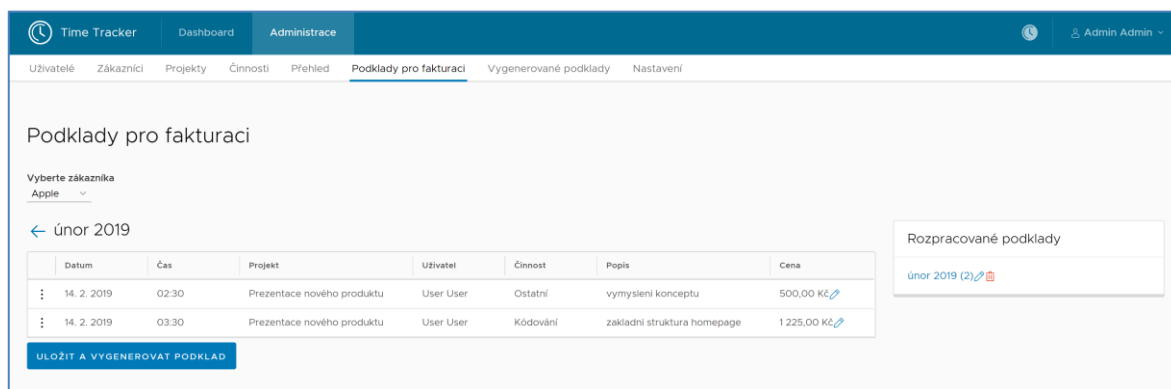
Nejdříve je nutné vytvořit alespoň jeden rozpracovaný podklad (celkově ale může být rozpracovaných podkladů libovolné množství). Poté je možné označit aktivity, které chce uživatel do podkladu vložit a tlačítkem „do podkladu“ nad tabulkou zvolí daný rozpracovaný podklad. Aktivity se do něj okamžitě přiřadí (projeví se v pravém sloupci navýšením počtu aktivit v podkladu) a odeberou se ze seznamu. Aktivity je samozřejmě možné z rozpracovaného podkladu kdykoliv odebrat, případně odstranit celý rozpracovaný podklad (všechny v něm obsažené aktivity se vrátí zpět do přehledu).



Obrázek 15: Podklady pro fakturaci

Po rozkliknutí rozpracovaného podkladu v pravém sloupci se místo seznamu nevyfakturovaných aktivit zobrazí pouze aktivity přiřazené danému podkladu. U každé aktivity je možné upravit výslednou fakturovanou částku.

Kliknutím na tlačítko „uložit a vygenerovat podklad“ a vyplnění názvu se podklad potvrdí a už v něm nebude možné dělat žádné změny. Odebere se z rozpracovaných podkladů a v modulu „Podklady pro fakturaci“ nebude vůbec dostupný. Přesune se do modulu „Vygenerované podklady“.



Obrázek 16: Rozpracovaný podklad

6.4.7 Vygenerované podklady

V tomto modulu se nacházejí všechny vygenerované podklady z modulu „Podklady pro fakturaci“. Není s nimi možné už nijak manipulovat, pouze lze stáhnout podklad jako soubor PDF s aktivitami členěnými podle projektů.

Time Tracker			
Dashboard			
Administrace			
Uživatelé Zákazníci Projekty Činnosti Přehled Podklady pro fakturaci Vygenerované podklady Nastavení			
Vygenerované podklady			
Název	Zákazník	Vytvořeno	Stáhnout
únor 2019	Apple	10. 3. 2019	Stáhnout
únor 2019	Google	14. 2. 2019	Stáhnout

Obrázek 17: Seznam vygenerovaných podkladů

Time Tracker

Podklad pro fakturaci

Klient: BBB

Vytvořeno: 15. 1. 2019

NOVEJ

TEST	01:00	300.00
Celkem	01:00	300.00

Jsem projekt

kodoval jsem, fakt.	01:00	100.00
Uz mam opravdickej cas!	01:00	100.00
Celkem	02:00	200.00

Jsem dalsi projekt

?	01:00	115.00
ert	00:15	25.00
jjj	01:00	100.00
Celkem	02:15	240.00

Součet

05:15 hodin

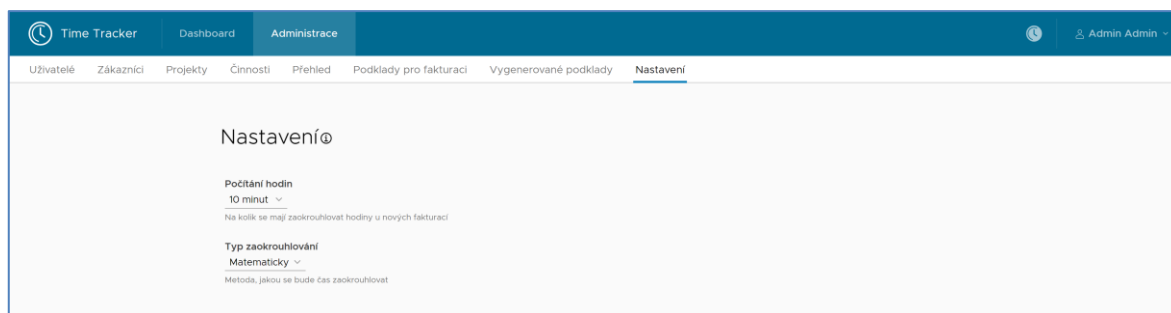
Součet

740.00 Kč

Obrázek 18: Ukázka vygenerovaného PDF

6.4.8 Nastavení

Zde se nastavují hlavní parametry pro fungování aplikace. Je to typ počítání hodin a typ zaokrouhlování. Hodnoty se ukládají automaticky při změně.



Obrázek 19: Modul nastavení

6.4.8.1 Počítání hodin

Na vygenerovaných podkladech se zobrazuje čas pouze zaokrouhleně. Je možné zaokrouhlovat na následující časové úseky:

- Minuta
- Pět minut
- Deset minut
- Čtvrt hodiny
- Půl hodiny
- Hodina

6.4.8.2 Typ zaokrouhlování

Určuje, jakým způsobem se bude čas zaokrouhlovat. Dostupné jsou následující možnosti:

- Matematicky (podle pravidel matematického zaokrouhlování)
- Vše nahoru (vše zaokrouhlí na nejbližší vyšší hodnotu)
- Vše dolů (vše zaokrouhlí na nejbližší nižší hodnotu)

Závěr

Hlavním cílem projektu bylo vytvořit jednoduchou aplikaci pro evidenci práce a generování podkladů pro fakturaci, která se bude reálně používat. A to se podařilo. Aplikace je v tuto dobu nasazená ve firmě na testovacím provozu a brzy půjde na produkci. Už nyní jsou vymyšleny plány do budoucna, v přípravě je například ještě jednodušší evidence práce pomocí stopek, nebo napojení na fakturační aplikaci Fakturoid. Plánujeme také mobilní aplikaci nebo například aplikaci pro zákazníky, ve které se budou moci podívat na průběh práce na jejich projektech.

Projekt měl velký význam nejen pro firmu, ale také pro mě. Naučil jsem se pracovat s novou formou ukládání dat pomocí NoSQL databáze, zdokonalil jsem se v tvorbě webových aplikací v JavaScriptu. Zejména jsem se velmi zlepšil v práci s JavaScriptovým frameworkem Angular a naučil jsem se základní principy komunikace mezi serverem s klientem a její zabezpečení.

Myslím, že jsem projektem také dokázal, že lze napsat kompletní aplikaci v jednom programovacím jazyku – v JavaScriptu. Vyhnul jsem se tak kombinování několika naprosto odlišných jazyků dohromady, například velmi běžné kombinaci PHP – MySQL – JS. A řekl bych, že díky tomu se vývoj zjednodušil, jednotlivé části spolu lépe komunikují a odpadl problém s řešením různorodosti jazyků.

Seznam přístupových údajů

URL adresa webu:

Úroveň oprávnění	Přihlašovací jméno	Heslo
Administrátor	admin	r4dM8MVinWv8jQJz
Zaměstnanec	user	P66XgAYU6MPucpXa

Seznam obrázků

Obrázek 1: Toggle (zdroj: vlastní)	12
Obrázek 2: TimeDoctor (zdroj: alternativeto.net).....	13
Obrázek 3: Clockify (zdroj: alternativeto.net)	14
Obrázek 4: Use Case diagram	16
Obrázek 5: struktura databáze	17
Obrázek 6: Struktura backendu	22
Obrázek 7: Struktura kódu frontentu	26
Obrázek 8: Přihlašovací obrazovka.....	33
Obrázek 9: Dashboard.....	34
Obrázek 10: Dialog pro evidenci práce.....	35
Obrázek 11: Administrační rozcestník.....	35
Obrázek 12: Správa uživatelů	36
Obrázek 13: Správa projektů.....	37
Obrázek 14: Přehled aktivit.....	38
Obrázek 15: Podklady pro fakturaci	39
Obrázek 16: Rozpracovaný podklad	39
Obrázek 17: Seznam vygenerovaných podkladů	40
Obrázek 18: Ukázka vygenerovaného PDF	40
Obrázek 19: Modul nastavení	41

Zdroje

1. Angular (application platform). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-02-11]. Dostupné z: [https://en.wikipedia.org/wiki/Angular_\(application_platform\)](https://en.wikipedia.org/wiki/Angular_(application_platform))
2. Express.js. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-02-11]. Dostupné z: <https://en.wikipedia.org/wiki/Express.js>
3. JavaScript. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-02-11]. Dostupné z: <https://en.wikipedia.org/wiki/JavaScript>
4. MongoDB. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-02-11]. Dostupné z: <https://en.wikipedia.org/wiki/MongoDB>
5. *Mongoose ODM* [online]. [cit. 2019-02-11]. Dostupné z: <https://mongoosejs.com/>
6. Node.js. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-02-11]. Dostupné z: <https://en.wikipedia.org/wiki/Node.js>
7. Npm (software). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-02-11]. Dostupné z: [https://en.wikipedia.org/wiki/Npm_\(software\)](https://en.wikipedia.org/wiki/Npm_(software))
8. *Pug* [online]. [cit. 2019-02-11]. Dostupné z: <https://pugjs.org/api/getting-started.html>
9. *Stylus Lang* [online]. [cit. 2019-02-11]. Dostupné z: <http://stylus-lang.com/>
10. TypeScript. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-02-11]. Dostupné z: <https://en.wikipedia.org/wiki/TypeScript>

Přílohy

Příloha 1 - Zdrojový kód aplikace na USB disku.