SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Tím 16 - MI16

Inteligentný importér verejných datasetov

Dokumentácia k riadeniu

Vedúci tímu:
Ing. Jakub Šimko, PhD.

Členovia tímu:
Bc. Ladislav Bari
Bc. Jaroslav Schvantner
Bc. Adam Ševčík
Bc. Adam Talian
Bc. Filip Varga
Bc. Milan Vaško
Bc. Jana Vrabľová

Obsah

Úvod

Tento dokument obsahuje dokumentáciu k riadeniu projektu na predmete Tímový projekt v zimnom semestri ak. roka 2018/19. Dokument obsahuje popis rolí členov tímu, aplikácia použitých manažmentov pri riadení, sumarizáciu šprintov a retrospektívu na doterajšiu prácu na projekte a dosiahnuté výsledky.
Prvá kapitola poskytuje sumarizáciu uvedeného obsahu.
V druhej kapitole popisujeme role jednotlivých členov tímu, ich zodpovednosti a oblasti expertízy počas práce na projekte.
Tretia kapitola poskytuje prehľad aplikovaných manažmentov - dokumentácie, plánovania, komunikácie, verziovania, kontroly kvality a testovania.
Štvrtá kapitola obsahuje zoznam ukončených šprintov v priebehu semestra, spoločne s popisom realizovaných používateľských príbehov.
Nasleduje piata kapitola spolu s globálnou retrospektívou a šiesta kapitola s motivačným dokumentom.
V siedmej kapitole je zoznam použitých metodík definovaných tímom v priebehu šprintov.
Ako prílohy sú priložené ešte k dokumentácií export evidencie úloh a stav webového sídla ku dňu odovzdania dokumentácie.

1. Roly členov tímu

Charakter nášho tímu je predovšetkým založený na agilite, preto sa snažíme aby ani jeden z členov tímu nebol špecializovaný výlučne na jednu oblasť expertízy. V priebehu trvania tímového projektu chceme naďalej pokračovať v snahe distibuovať vedomosti (ideálne) medzi všetkých členov tímu. Súčasťou toho je aj rotovanie funkcie scrum mastera po určitom počte šprintov.

Bc. Ladislav Bari
- backend developer
- zloženie formátu RDF, export načítaných dát do RDF

Bc. Jaroslav Schvantner
- tvorba webového sídla, v priebehu druhého šprintu zanechané štúdium

Bc. Adam Ševčík
- frontend developer
- grafický dizajnér

Bc. Adam Talian
- fullstack developer
- správa webového sídla tímu, staging a production serverov, deploy, code reviews
- písanie zápisníc z tímových stretnutí

Bc. Filip Varga
- backend developer

- bývalý scrum master, RDF formát, seléniové testy

Bc. Milan Vaško

- fullstack developer
- správa repozitára (Gitlab), code reviews
- písanie zápisníc z tímových stretnutí

Bc. Jana Vrabľová

- fullstack developer
- aktuálne scrum master
- správa tímovej wiki

2. Aplikácia manažmentov

V našom tíme sme k viacerým oblastiam riadenia vytvorili metodiky práce, ktoré sú podrobnejšie opísané v kapitole Metodiky.

2.1 Manažment dokumentácie

Pri každom tímovom stretnutí jeden prípadne dvaja členovia tímu robia zápisnicu, v ktorej je zaznamenaný priebeh stretnutia. Jednotlivé zápisnice sú dostupné na webovom sídle nášho tímu. Dokumentované sú aj metodiky, ktoré počas procesu riadenia tímového projektu vznikajú.

2.2 Manažment úloh

Na manažment úloh používame nástroj JIRA, kde sú uložené user stories a na ich základe vytvárame sub-tasky, ktoré na začiatku šprintu rozdelíme medzi členov tímu. Viac o manažmente úloh a o používanej metodike, je možné nájsť v príslušnej časti nižšie.

2.3 Manažment komunikácie

Komunikácia tímu prebieha počas stretnutí a na stand-upoch, pričom ak nie je možné osobné stretnutie, tak sú organizované prostredníctvom Slacku. Stand-upy mávame každý druhý deň. Komunikáciu na Slacku máme rozdelenú na viaceré kanály, ktoré sú zamerané na konkrétne diskusie alebo na integráciu s Jirou.

2.4 Manažment verziovania

Na verziovanie kódu používame Gitlab, ku ktorému máme napísanú metodiku (kapitola Metodiky).

2.5 Manažment nasadzovania

Naše riešenie je nasadené na dvoch serveroch, z ktorých je jeden vývojový a druhý produkčný. Príslušnú metodiku je možné nájsť v kapitole Metodiky.

2.6 Manažment vývoja a prehliadania zdrojového kódu

Pre vývoj a prehliadanie zdrojového kódu máme jasne napísanú metodiku, ktorú je možné nájsť v príslušnej sekcii kapitoly Metodiky.

3. Sumarizácie šprintov

V tejto časti stručne sumarizujeme uskutočnené šprinty.

3.1 Šprint 1 - “Dr. No”

V tomto šprinte sme mali zahrnuté rozbehanie riešenia minuloročného tímu (lokálne aj na serveri). Okrem toho sme spracovali tri user stories. Zistili sme, že by sme sa mali riadiť dizajn manuálom pre štátne weby (minuloročné riešenie tomu vôbec nezodpovedá, nakoľko bol tento manuál vydaný relatívne nedávno). Identifikovali sme potrebu väčšieho refaktoru kódu.

Riešené user stories

IM-1 Ako úradník chcem po spracovaní súboru stiahnuť výsledok ako súbor s RDF (ale s fake dátami)

- pridanie nového odkazu na frontend na stiahnutie RDF
- backend po použití odkazu vráti RDF s napevno danými údajmi

Stav na konci šprintu:

- požadovaná funkcionalita bola implementovaná, otestovaná a nasadená

IM-13 Rozbehať si riešenie lokálne (u všetkých)

- infraštruktúrna úloha, každý člen tímu musel rozchodiť technológie, v ktorých bol existujúci projekt vytvorený (Java, Angular, Docker)

Stav na konci šprintu:

- každý člen tímu bol schopný vyvíjať na svojom počítači

IM-14 Rozbehať si riešenie v cloude

- infraštruktúrna úloha, bolo potrebné vytvoriť server, na ktorom bude bežať inštancia našej webovej aplikácie

Stav na konci šprintu:

- web bol dostupný a použiteľný po zadaní IP adresy servera s využitím SSH tunelu

IM-17 Ako úradník chcem pri prechode kurzorom nad alternatívou zobraziť tooltip, čo je to za alternatívu, aby som lepšie vedel, čo je to za atribút a či je pre daný stĺpec vhodný

- v našej webovej stránke sa pracuje s tabuľkami údajov, pričom je potrebné pri každom stĺpci určiť jeho typ
- o tomto type je možné zobraziť viac informácií, ktoré mali byť dostupné práve cez tooltip

Stav na konci šprintu:

- pri prechode myšou nad ikonou lupy sa zobrazil tooltip s dodatočnými informáciami

IM-9 Ako úradník chcem po nahratí súboru rozhodnúť, ktorá karta v Excel súbore sa má spracovať, aby sa nespracovali tie, ktoré netreba

- vzhľadom k tomu, že Excel podporuje viachárkové súbory, je potrebné vyzvať používateľa, aby si vybral hárok, ktorý ho zaujíma

Stav na konci šprintu:

- user story sa nepodarilo dokončiť celé
- stihla sa dokončiť backendová funkcionalita vyťahovania názvov hárkov z Excel súboru
- nestihol sa však dorobiť frontend - hlavnými problémami bolo pravdepodobne podcenenie zložitosti frontendového kódu a neskorý začiatok prác na tasku
- taktiež sme prišli k záveru, že sme si zle rozdelili tasky - príliš medzi sebou súviseli a nedalo sa tak na nich robiť efektívne
- v dôsledku tejto úlohy sme počas retrospektívy zaviedli nové pravidlá:
    - každý task musí byť v prvom týždni šprintu aspoň začatý
    - treba si tasky lepšie rozdeliť - tak, aby medzi sebou čo najmenej súviseli
    - pri veľkom tasku je vhodné využiť párové programovanie

3.2 Šprint 2 - “You only live twice”

V tomto šprinte sme sa venovali refaktoru backendu, oboznamovaniu sa s formátom RDF, začali sme s prípravou funkcionality pre generovanie výsledného RDF súboru a na testovacie účely sme si vytvorili staging server. V tomto šprinte sme prišli o jedného človeka z tímu, ktorý zanechal štúdium.

Riešené user stories

IM-25 Rozbehať v cloude bez tunelov

Vtedajší stav:

- riešenie beží v cloude, kde je povolená HTTP prevádzka
- je možné sa po zadaní IP servera pripojiť na web
- pre funkčnosť (volanie backendu) je potrebné mať nastavené lokálne tunely - t.j. presmerovanie prevádzky z localhost na server
- v rámci FE je environment.ts a environment.prod.ts - zrejme sa to nepoužíva…

Stav na konci šprintu:

- napasovanie prostredia (environment) na frontende
- úprava build skriptu, aby využíval toto prostredie

IM-28 Refaktor handler-ov na backend-e

Vtedajší stav

- pôvodné handler-y nevyužívali vlastnosti použitého SpringBoot-u
- neprehľadný zdrojový kód (veľa try-catch blokov, zlé formátovanie)

Stav na konci šprintu:

- prepracovanie handlerov na SpringBoot služby
- vytvorenie centrálneho ErrorHandler-a, dôsledkom bude prehľadnejší kód
- naformátovanie zdrojových kódov podľa jednotného formátu
- pridaná debug úroveň logovania chýb

IM-32 Analýza - vyrobiť ručne príklady súborov RDF k definovaným príkladom

Požiadavky:

- 5 dvojíc súborov (rdf + xlsx) pre rozdielne entity
- očakáva sa, že výstupom budú nové story v backlog-u, ktoré budú smerovať k transformácii, ale aj k zisťovaniu ďalších informácií od používateľa.

Stav na konci šprintu::

- v tejto story boli vytvorené tabuľky vo formáte xlsx a k nim príslušné súbory vo formáte rdf
- všetky rdf súbory boli úspešne validované pomocou validátora

IM-33 Ako úradník chcem zadať akú entitu reprezentuje jeden riadok tabuľky aby bolo možné exportovať výsledné RDF

Požiadavky:

- pridanie dropdown-u, ktorý bude obsahovať zoznam všetkých možností (čo za entitu to môže byť), s tým, že používateľ bude musieť vybrať jednu z možností pre pokračovanie.
- Možnosti budú načítané z CM.
- Vybraná možnosť sa musí poslať na backend.

Stav na konci šprintu::

- podľa požiadaviek pridaný dropdown
- údaje načítavané z centrálneho modelu uloženom v ElasticSearch
- vybraná možnosť sa posiela na backend

IM-42 Vytvoriť a nastaviť development (staging) server (aj CI)

Požiadavky:

- možnosť testovať zmeny pred nasadením ostrej verzie
- nutnosť vytvoriť prostredie, ktoré je čo najbližšie produkcii a zároveň je prijateľné, aby obsahovalo nedokončenú funkcionalitu

Stav na konci šprintu::

- vytvorenie nového serveru
- nastavenie CI na Gitlabe na automatický build a deploy zmien pushovaných do Git repozitáru

IM-09 Ako úradník chcem po nahratí súboru rozhodnúť, ktorá karta v excel súbore sa má spracovať, aby sa nespracovali tie, ktoré netreba

Požiadavky:

- potreba dokončiť user story, ktorá nebola dokončená v minulom šprinte
- chýbalo prepojenie backendovej časti s frontendom

Stav na konci šprintu::

- párové programovanie
- vhodná úprava posielaných dát
- refaktor určitých častí backendu kvôli potrebám frontendu

3.3 Šprint 3 - “Live and let die”

Šprint za 12 story points z toho získaných 12 story points.
Hlavnou náplňou tohto šprintu je analýza frontendu. Súčasný zdrojový kód nám spôsobuje problémy, chceli by ho nejako zmeniť - analyzujeme možnosti (refaktor súčasného riešenia vs. napísanie nanovo). Nové refaktorované riešenie bude v súlade s dizajn manuálom pre štátne weby. Okrem toho pracujeme na spoznávaní formátu RDF a oprave niektorých chýb, ktoré sme doteraz identifikovali (používanie diakritiky, vytvorenie názvu nového stĺpca).

Riešené user stories

IM-9 Ako úradník chcem po nahratí súboru rozhodnúť, ktorá karta v excel súbore sa má spracovať, aby sa nespracovali tie, ktoré netreba

- v tomto šprinte sa nám podarilo dokončiť túto úlohu, ktorú sme riešili od prvého šprintu

IM-27 Opraviť chybu pri vytvorení nového názvu stĺpca

- chyba sa vyskytovala, ak používateľ zadal možnosť vyhľadávania v CM alebo pri možnosti *Vytvoriť nový názov stĺpca*
- v UI sa prejavila hláškou "*Chyba p.Undefined error* "

Stav na konci šprintu:

- Chyba spočívala v púšťaní agregačného dopytu do Elasticsearchu nad field-om typu text. Podľa dokumentácie môžeme púšťať agregačné dopyty len nad field-om typu 'keyword'. Vyriešené pridaním field-u typu keyword s duplikovanými dátami.

IM-30 Aktualizovať dokumentáciu na wiki

- na tímovej wiki nebola aktualizovaná dokumentácia, rozbehávanie technológií boli naposledy aktualizované minuloročným tímom
- tiež bolo potrebné zdokumentovať rôzne metodiky, ktoré sme si ako tím zaviedli

Stav na konci šprintu:

- boli spísané metodiky, ktorými sa riadime, a tiež boli aktualizované súbory, týkajúce sa technológií, ktoré v projekte používame

IM-45 Vo vyhľadávaní v celom modeli musí fungovať aj dopyt s diakritikou/veľkosť písmen / medzery

- vyhľadávanie alternatívneho názvu stĺpca v CM nefungovalo správne, pri použití medzery, slov s diakritikou alebo pri veľkých písmenách neboli nájdené alternatívy k názvu stĺpca v tabuľke

Stav na konci šprintu:

- problém sa odstránil tak, že pri vyhľadávaní sa výraz rozdelí na samostatné slová, odstráni sa diakritika, a veľkosť písmen sa zjednotí na lowercase
- pri riešení tohto problému sa vytvoril nový task, ktorý vyplynul z toho, že vyhľadávanie podľa zadaných kľúčových slov neposkytuje dostatočne relevantné výsledky, preto bol vytvorený task IM-67 Lematizovanie query pri vyhľadávaní stĺpcov

IM-53 Vytvorenie príkladov súborov RDF

- táto user story nadväzuje na analýzy z predošlých šprintov, kde sme analyzovali štruktúru RDF súborov
- na základe analýzy bolo potrebné pripraviť vzorové súbory RDF, na oboznámenie sa s týmto formátom

Stav na konci šprintu:

- boli vytvorené RDF súbory, ktoré budú v ďalšej práci na projekte slúžiť ako vzor pre generovanie RDF súborov, ktoré budú nahraté na stránku v inom formáte

IM-54 Analyzovať možnosti refaktoringu frontendu

- na základe stavu frontendu z minulého roka bolo potrebné analyzovať možnosti refaktoringu, ktorý súvisí aj s implementáciou oficiálneho dizajn manuálu
- v rámci tejto user story bolo potrebné analyzovať súčasnú štruktúru frontendu v Angulari a tiež analyzovať prípadné použitie alternatívnej technológie, kde sme zvažovali najmä Elm.

Stav na konci šprintu:

- na základe analýz a diskusie počas tímového stretnutia sme dospeli k záveru, že bude vhodnejšie vytvoriť novú verziu webovej stránky v aktualizovanej verzii Angularu, kde bude možné použiť časť funkcionality existujúceho riešenia

IM-55 Vytvorenie metodiky pre kód

- pre zjednotenie štýlu písania zdrojového kódu bolo potrebné vytvoriť dokument, ktorý to opisuje

Stav na konci šprintu:

- bola vytvorená metodika “Štýl programovania”, ktorá je dostupná na tímovej wiki

3.4 Šprint 4 - “For your eyes only”

Šprint za 16 story points, z toho získaných 0 story points.
Tento šprint bol venovaný dvom dôležitým častiam. Prvá predstavovala kompletný refaktoring front-endu a jeho úpravu podľa zverejneného jednotného dizajn manuálu elektronických služieb Slovenskej republiky. Druhým zameraním tohto šprintu bola implementácia RDF generátora, vďaka ktorému je možné spracovaný súbor konvertovať na RDF súbor a následne ho stiahnuť.

Riešené user stories

IM-26 Implementácia novej štruktúry front-endu

- bolo nutné urobiť rozsiahly refaktor vtedajšieho frontendového kódu, ktorý nespĺňal naše štandardy kvalitného kódu a tiež na viacerých miestach úplne vynechával spracovanie chýb a tiež nevyužíval viaceré funkcie jazyka TypeScript zvyšujúce robustnosť kódu

Stav na konci šprintu:

- user story zostala v pokročilom štádiu rozpracovania
- zistili sme, že danú user story sme veľmi podhodnotili a vôbec nemala ísť do šprintu celá
- veľká časť refaktoru bola hotová, ale stále ostávalo veľa práce

IM-63 Ako úradník chcem po spracovaní hárku vygenerovať súbor formátu RDF

- cieľom bolo vytvoriť generátor súborov RDF, ktorý by bol schopný vygenerovať RDF vo forme príkladov vytvorených v predchádzajúcom šprinte, ale s reálnymi dátami
- malo ísť o prvú základnú verziu, na ktorej sa malo neskôr iterovať

Stav na konci šprintu:

- backend časť funkčná
- chýba prepojenie s frontendom

IM-65 Implementácia front-endu podľa dizajn manuálu

- v rámci refaktoru frontendu sme sa rozhodli zároveň implementovať do projektu dizajnové princípy stanovené dizajn manuálom štátnych webov (https://idsk-elements.herokuapp.com/)

Stav na konci šprintu:

- podobný stav ako pri refaktore - veľká časť hotová, ale trochu práce ešte zostalo

IM-68 Dokumentácia

- bolo nutné dokončiť dokumentáciu v zmysle požiadaviek na stránke predmetu

Stav na konci šprintu:

- odovzdaná bola najlepšia verzia, akú sme boli schopní vyprodukovať, ale do finálneho odovzdania sme dokumentáciu chceli ešte upraviť a zrevidovať, preto sme túto úlohu nepovažovali za hotovú

3.4 Šprint 5 - “Never Say Never Again”

Šprint za 16 story points, z toho získaných 16 story points.
Keďže v predošlom šprinte sme neukončili úspešne ani jednu user story, tak v tomto šprinte sme sa rozhodli riešiť opätovne všetky a dokončiť ich. Jednalo sa o refaktoring frontendu, dokončenie RDF generátora a dokumentácie. V tomto šprinte boli všetky user stories dotiahnuté do úspešného konca a vytvorené nové user stories do backlogu.

Riešené user stories

IM-26 Implementácia novej štruktúry front-endu

- bolo nutné urobiť rozsiahly refaktor vtedajšieho frontendového kódu, ktorý nespĺňal naše štandardy kvalitného kódu a tiež na viacerých miestach úplne vynechával spracovanie chýb a tiež nevyužíval viaceré funkcie jazyka TypeScript zvyšujúce robustnosť kódu

Stav na konci šprintu:

- frontend bol úspešne refaktorovaný
- boli preň vytvorené nové testy

IM-63 Ako úradník chcem po spracovaní hárku vygenerovať súbor formátu RDF

- cieľom bolo vytvoriť generátor súborov RDF, ktorý by bol schopný vygenerovať RDF vo forme príkladov vytvorených v predchádzajúcom šprinte, ale s reálnymi dátami
- malo ísť o prvú základnú verziu, na ktorej sa malo neskôr iterovať

Stav na konci šprintu:

- generátor funguje korektne, komunikuje s frontendom a generuje súbor
- boli identifikované nové problémy, ktoré je nutné zapracovať

IM-65 Implementácia front-endu podľa dizajn manuálu

- v rámci refaktoru frontendu sme sa rozhodli zároveň implementovať do projektu dizajnové princípy stanovené dizajn manuálom štátnych webov (https://idsk-elements.herokuapp.com/)

Stav na konci šprintu:

- až na drobné kozmetické úpravy je front-end úspešne aktualizovaný a zodpovedá štandardom stanoveným dizajn manuálom

IM-68 Dokumentácia

- bolo nutné dokončiť dokumentáciu v zmysle požiadaviek na stránke predmetu

Stav na konci šprintu:

- dokumentácia bola finalizovaná, vygenerovaná a odoslané do AIS-u

4. Globálna retrospektíva

Za prvé štyri šprinty sa nám podarilo naimplementovať časť funkcionality, ktorá je hodnotná pre zákazníka (vedúci tímu). No najmä v prvom šprinte sme sa viac venovali rozbehávaniu

technológií. Prvé dva šprinty sme nemali ukončené všetky úlohy. Išlo najmä o zlé rozdelenie user story na podúlohy, ktoré boli príliš závislé, čo nás výrazne spomalilo. Tiež sme nedostatočne odhadli náročnosť niektorých úloh. Tretí šprint bol oveľa úspešnejší a podarilo sa nám v ňom splniť všetky úlohy. Štvrtý šprint nebol napriek veľkej snahe úspešný, čo bolo spôsobené najmä časovou tiesňou, spôsobenou študijnými povinnosťami. Tiež je možné povedať, že nebolo správnym rozhodnutím, vziať do šprintu dve stories, z ktorých obe boli ohodnotené na 8 story pointov.

Po každom šprinte sme si na základe retrospektívy vytvorili, upravili alebo prípadne odstránili pravidlá, ktorými sa riadil tím. Snažili sme sa, aby sme v tíme spolupracovali s čo najväčšou efektivitou. Aktuálne pravidlá, ktorými sa riadime sú podrobnejšie popísané v časti *Metodika práce.*

Počas trvania štyroch šprintov, sme sa stretli s mnohými problémami, ktoré sme sa snažili v čo najväčšej miere vyriešiť. Medzi tieto problémy môžeme zaradiť napríklad vznik závislostí, medzi jednotlivými sub-taskami, ktoré sme vyriešili párovým programovaním. Ďalším veľkým problém, kvôli ktorému zlyhal náš prvý šprint bola nízka miera komunikácie a s tým spojený malý prehľad o práci ostatných kolegov. Riešením tohto problému bolo zriadenie pravidelnejších standupov, konkrétne každý druhý deň. Ďalšími problémami boli neskoré začatie s prácou na úlohách, čo sme vyriešili pravidlom, že všetky úlohy musia byť rozpracované počas prvého týždňa a nejednotné písanie zdrojového kódu, čoho riešením bolo vytvorenie metodiky, na jeho písanie. Keďže sme preberali rozbehnutý projekt, problémom bola aj neznalosť problematiky a technológií. Riešením tohto problému bolo vytvorenie analytických úloh, ktorých úlohou bolo zoznámenie sa s danou oblasťou, problematikou a používanými technológiami. Ako nedávny problém sa ukázali neočakávané chyby pri nasadzovaní riešenia na server. Riešením je teda pravidelné priebežné nasadzovanie riešenia na server.

5. Motivačný dokument

Uvedený motivačný dokument bol predložený pri výberoch tímov na témy.

Predstavenie tímu

E-mailový kontakt na tím: tp.tim16@gmail.com

- Náš tím je tvorený energickým a taktiež flexibilným kolektívom so zameraním na rôzne oblasti informatiky, vrátane ľudí s určitými skúsenosťami z praxe. Piati členovia študujú odbor ISS2, dvaja IT2, čím vytvárame vyvážený pomer zameraní na väčšinu projektov.
- Všetci členovia tímu ovládajú na pokročilej úrovni:
  - C, C++, Java, Python (vedomosti z nich boli získané najmä počas bakalárskeho

štúdia na FIIT, ale u viacerých aj praxou),
    - práca s SQL databázami - PostgreSQL (opätovne štúdium na FIIT),
    - zručnosti v tvorbe webstránok využitím HTML, CSS, JS (prevažne vlastná prax),
    - XML a JSON (prevažne štúdium na FIIT),
    - Git.
- Časť tímu dokáže ponúknuť aj nasledovné schopnosti:
    - C#, .NET (vlastné skúsenosti a prax),
    - JavaEE (skúsenosti z predmetu VAsVA),
    - Javascript, AngularJS (vlastné skúsenosti),
    - Assemblery (skúsenosti z predmetu SPAASM),
    - Funkcionálne a logické programovanie - Lisp, Prolog (skúsenosti z predmetu FLP) **.**
- Výraznejšie špecializácie členov:
    - Milan Vaško: najvšestrannejší programátor, má skúsenosti s Go, Rust, Elm, skriptovaním v Bashi, Unity, či Lua a Docker, s ktorými pracoval na svojej BP.
    - Adam Talian: skúsenosti najmä s vývojom v C#. Má skúsenosti s eye-trackingom s ktorým sa stretol v rámci svojej BP, ktorej sa následne následne venoval ďalej počas leta pod vedením vedúceho.
    - Adam Ševčík: Pracoval s technológiami ako WebGL, ThreeJS, NodeJS alebo noSQL databázou MongoDB a mimo iného ovláda PHP, Laravel alebo Bootstrap
    - Ladislav Bari a Jaroslav Schvantner: obaja sa zaujímajú o počítačovú grafiku a počítačové videnie, počas najbližšieho semestra sa budú venovať predmetu Spracovanie obrazu, grafika a multimédia, Ladislav už pracoval s Tensorflow
    - Jana Vrabľová: na svojej bakalárskej práci sa zoznámila s jazykom SystemVerilog a bude navštevovať predmet Základy kryptografie
    - Filip Varga: zaoberá sa automatizačnými procesmi v praxi v biznise pomocou jazykov VBA, AutoIt a využíva Selenium v Pythone, bude navštevovať predmet Spracovanie informácií v podnikaní a verejnej správe
- Počas bakalárskeho štúdia sme sa venovali strojovému učeniu na predmete Inteligentná analýza údajov, jazyku XML na predmete Webové publikovanie, alebo tvorbe softvéru na Aplikačnom programovaní v C++ a Vývoji aplikácií s viacvrstvovou architektúrou
- Počas nasledujúcich dvoch semestrov budú členovia nášho tímu študovať spoločne najmä predmety: Objektovo-orientovaná analýza softvéru, Počítačové videnie, Štatistické metódy vyhodnocovania experimentov a Vizualizácia dát
- Niektorí členovia obohatia znalosti tímu o vedomosti v oblasti umelej inteligencie z predmetu Neurónové siete, testovania z predmetov Kvalita programových a informačných systémov ,či Testovanie softvéru alebo predmety z odboru IT2 ako: Komunikačné služby a siete, Vnorené systémy a Bezpečnosť v internete.
- Jeden z členov nášho tímu má pracovné skúsenosti s krátkodobým vedením menšieho tímu v roli team leadera, avšak nepracovali na vývoji spoločného softvéru, jednalo sa predovšetkým o analýzu dát, či menšie individuálne projekty
- Okrem už dosiahnutých vedomostí a zručností je tím pripravený čeliť výzvam v oblastiach tímovej spolupráce a komunikácie, a taktiež učeniu sa práci s novými technológiami.

- Rôzne zamerania členov tímu prinášajú zaujímavú možnosť poskytnutia rôznych uhlov pohľadov na daný problém

Motivácia k téme 2 - Inteligentný importér verejných dát [Importer]

Čo nás zaujalo na tejto téme:

- previazanosť s praxou
- príležitosť zlepšenia súčasného stavu IT vo verejnej správe
- podieľanie sa na vývoji ‘praktického’ softvéru, ktorý má potenciál nájsť si väčšie uplatnenie
- získanie skúseností s prácou s veľkým množstvom informácií
- nadviazanie na doterajšie získané znalosti z oblasti strojového učenia a zároveň aj tvorby softvéru

Aké máme vedomosti/schopnosti užitočné pri tejto téme:

- ovládame Javu, prácu s Git-om, niektorí z nás už majú aj skúsenosti s prácou s Dockerom aj AngularJS
- viacerí z nás absolvovali predmet Inteligentná analýza údajov, na ktorý nadväzuje Objavovanie znalostí
- v rozvrhoch máme zvolené predmety súvisiace s danou témou ako Objavovanie znalostí, Neurónové siete, Štatistické metódy vyhodnocovanie experimentov a Spracovanie informácií v podnikaní a verejnej správe, čo môže taktiež prispieť k zefektívneniu vypracovania tejto témy
- niektorí členovia tímu pracovali s jazykom XML na predmete Webové publikovanie počas bakalárskeho štúdia, čo je vhodný (a mal by byť aj často používaný) formát pre štruktúrované dáta pre uchovávanie rôznych typov informácií a dokumentov poskytujúci rozsiahle možnosti validácie a transformácie

Príloha A - Poradie tém

2. Inteligentný importér verejných dát
10. IoT systém monitorovania osôb
3. Vyhľadávanie pomocou obrázkov
17. Analýza správania sa vozidiel v meste
18. Škola hrou vo virtuálnej realite

9. Vnímanie neviditeľného
6. Monitorovanie a vyhodnocovanie fyziologických procesov človeka
7. Vizualizácia softvéru vo virtuálnej a rozšírenej realite (Remake)
5. Podpora výskumu behaviorálnej biometrie
12. In-memory databáza s využitím GPU
15. Monitoring antisociálneho správania
1. Prostredie pre inteligentnú analýzu textov
16. Prostredie na vizualizáciu mikrogridu
8. Generátor 3D priestoru
14. Identifikácia entít - spracovanie textu
4. 3D UML, improved version
19. Automatické testovanie v prostredí Internetu vecí
11. Databanka otázok a úloh
13. 3D simulovaný robotický futbal
20. WiFi Funtoro

Príloha B - Rozvrh tímu

<table>
<tr><th></th><th>8:00 - 8:50</th><th>9:00 - 9:50</th><th>10:00 - 10:50</th><th>11:00 - 11:50</th><th>12:00 - 12:50</th><th>13:00 - 13:50</th><th>14:00 - 14:50</th><th>15:00 - 15:50</th><th>16:00 - 16:50</th><th>17:00 - 17:50</th><th>18:00 - 18:50</th><th>19:00 - 19:50</th><th>20:00 - 20:50</th><th>21:00 - 21:50</th></tr>
<tr><td rowspan="2">Pondelok</td><td colspan="2">Kvalita programových a informačných systémov - Varga, Talian</td><td colspan="2">Architektúra informačných systémov - Bari, Schvantner</td><td colspan="2">Softvérové jazyky - Talian, Vaško</td><td colspan="3">Tímové stretnutie s vedúcim - preferované</td><td colspan="2">Architektúra informačných systémov - Bari, Schvantner</td><td></td><td colspan="2">Tímové stretnutie</td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="3"></td><td colspan="2"></td><td colspan="2">Softvérové jazyky - Talian, Vaško</td><td></td></tr>
<tr><td rowspan="3">Utorok</td><td colspan="3">Spracovanie informácií v podnikaní a verejnej správe - Varga</td><td colspan="3">Spracovanie obrazu, grafika a multimédiá - Bari, Schvantner</td><td colspan="3" rowspan="2">Architektúra softvérových systémov - Bari, Schvantner, Varga, Talian, Vaško</td><td rowspan="2">VISS - Bari, Schvantner, Varga, Talian, Vaško</td><td colspan="2" rowspan="2">Tímový projekt I - Bari, Schvantner, Vrabľová, Ševčík, Varga, Talian, Vaško</td><td colspan="2">Architektúra softvérových systémov - Bari, Schvantner, Vaško, Talian</td><td></td></tr>
<tr><td colspan="2">Základy kryptografie - Vrabľová</td><td colspan="2"></td><td colspan="2"></td><td colspan="2">Architektúra počítačových systémov - Vrabľová</td><td></td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2">Výskum systémov počítačového inžinierstva - Vrabľová, Ševčík</td><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td rowspan="2">Streda</td><td colspan="3">Neurónové siete - Vaško</td><td colspan="3">Základy kryptografie - Vrabľová</td><td colspan="2">Distribúcia obsahu v internete - Vrabľová, Ševčík</td><td colspan="2">Manažment v tvorbe softvéru - Bari, Schvantner, Varga, Talian, Vaško</td><td colspan="3">Manažment v tvorbe softvéru - Bari, Schvantner, Varga, Talian, Vaško</td><td colspan="2">Spracovanie informácií v podnikaní a verejnej správe - Varga</td><td></td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td><td colspan="2">Architektúra počítačových systémov - Vrabľová, Ševčík</td><td colspan="3">Distribúcia obsahu v internete - Vrabľová, Ševčík</td><td colspan="2"></td><td></td></tr>
<tr><td rowspan="3">Štvrtok</td><td colspan="4">Komunikačné služby a siete - Vrabľová</td><td colspan="3">Kvalita programových a informačných systémov - Varga, Talian</td><td colspan="3">Architektúra počítačových systémov - Ševčík</td><td colspan="2">Spracovanie obrazu, grafika a multimédiá - Bari, Schvantner</td><td colspan="3">Vnorené systémy - Ševčík</td><td colspan="2">Tímové stretnutie</td><td></td></tr>
<tr><td colspan="2"></td><td colspan="3">Architektúra softvérových systémov - Varga</td><td colspan="2"></td><td colspan="2"></td><td colspan="2">Vnorené systémy - Ševčík</td><td colspan="3"></td><td colspan="2"></td><td></td></tr>
<tr><td colspan="2"></td><td colspan="3">Neurónové siete - Vaško</td><td colspan="2"></td><td colspan="2"></td><td></td><td></td><td colspan="3"></td><td colspan="2"></td><td></td></tr>
<tr><td>Piatok</td><td colspan="2"></td><td colspan="4">Komunikačné služby a siete - Vrabľová</td><td colspan="4">Tímové stretnutie s vedúcim - druhá možnosť</td><td></td><td></td><td colspan="3"></td><td colspan="2"></td><td></td></tr>
</table>

6. Metodiky

6.1 Metodika manažmentu úloh

Manažment úloh predstavuje jednu z najdôležitejších častí pri riadení tímového projektu. Cieľom tejto metodiky je sumarizácia pravidiel, ktoré sú využívané pri tomto manažmente. Táto metodika je určená pre všetkých členov tímu.

6.1.1 Zaužívané pojmy

Epic

Epic predstavuje veľkú časť funkcionality, ktorá sa skladá z viacerých stories. Epic je vo všeobecnosti väčší ako jeden šprint.

Story

Story, taktiež volaná aj User Story, je menšia požiadavka na funkcionalitu. Účelom story je vyjadrenie toho, ako určitá funkcionalita prinesie zákazníkovi biznis hodnotu. Story, by nemala byť väčšia ako jeden šprint. V prípade, že táto podmienka nie je splnená, je potrebné danú story rozdeliť na niekoľko menších stories. V šprinte sa story delí na sub-tasky. Story je ohodnotená tzv. story pointami. O ohodnotení rozhodujú členovia tímu pomocou planning pokru.

Sub-Task

Sub-task predstavuje úlohu, ktorá je časťou jednej User Story. Táto úloha musí byť počas šprintu splnená.

Backlog

Backlog predstavuje zoznam úloh (stories), ktoré je potrebné v projekte vykonať. Jednotlivé položky sú prioritizované (čím vyššie v zozname, tým majú vyššiu prioritu). Prioritizovanie jednotlivých úloh prebieha počas tzv. backlog groomingu.

6.1.2 Vytvorenie úloh

Jednotlivé úlohy sú vytvárané priebežne počas šprintov. Dané úlohy sú následne uložené v backlogu. Každá vytvorená úloha (story) je zaradená do príslušného epicu, ktorý tematicky zastrešuje danú story. Aby mohla byť story pripravená na výber do šprintu, je potrebné, aby spĺňa tzv. Definition of ready. V našom prípade, je tento súbor pravidiel, definovaný takto:

- Story musí byť zaradená do epicu.
- Story musí mať dostatočne jasný opis. V ňom má byť jasne definované kto, čo a prečo.
- Story musí mať popísaný aspoň jeden používateľský scenár.
- Tím musí byť oboznámený s problematikou danej story.
- Tím musí mať prehľad o daných technológiách, ktorých použitie, si story bude vyžadovať.
- Story má veľkosť maximálne jedného šprintu.
- Story musí mať svoj odhad pomocou story pointov.
- Story musí mať jasne stanovené akceptačné kritériá.

Ak sú všetky tieto pravidlá splnené, je možné úlohu vybrať do šprintu.

6.1.3 Výber úloh do šprintu

Jednotlivé úlohy, ktoré budú vykonávané v šprinte sa vyberajú z backlogu. Keďže je backlog prioritizovaný, úlohy, ktoré sú najvyššie, budú vybraté najskôr. Výber úloh je však ovplyvnený aj počtom story pointov, ktorými sú dané stories ohodnotené. Keďže toto ohodnotenie reflektuje náročnosť danej story, tím si vyberie jednotlivé stories, ktorých súčet story pointov, je schopný tím zvládnuť.

6.1.4 Vykonanie úloh v šprinte

Po úspešnom vybratí úloh do šprintu, je šprint odštartovaný. Jednotlivé vybrané stories sa následne delia na menšie sub-tasky. Toto delenie má na starosti tím, pričom sa berie ohľad na to aby sa dané sub-tasky neprekrývali. Ak takéto rozdelenie nie je možné, preferujeme párové programovanie. Vytvorené sub-tasky sú následne pridelené jednotlivým členom tímu, ktorí sú teda zodpovední za ich vykonanie. Počas šprintu má sub-task niekoľko stavov, ktoré sme si zadefinovali. Nimi sú:

- To Do - úloha je pripravená na riešenie
- In Progress - pridelená osoba začala pracovať na úlohe
- Ready to Review - pridelená osoba dokončila prácu a riešenieje pripravené na kontrolu
- In Review - prebieha kontrola riešenia
- Ready To Deploy - riešenie bolo skontrolované a je pripravené na nasadenie
- Ready To Accept - riešenie je pripravené na akceptáciu vlastníkom projektu
- Done - riešenie spĺňa definition of done, úloha je splnená

Ako je spomínané, pred tým, ako je úloha presunutá do stavu Done, je potrebné, aby splnila tzv. Definition of done. V našom prípade je stanovené takto:

- Úloha musí byť implementovaná

- Implementácia je overaná testami (JUnit testy)
- Implementácia úspešne prešla code review
- Sú splnené akceptačné kritériá
- Implementácia je nasadená na “staging” servery
- Riešenie schválené vlastníkom projektu

Každá úloha má svoj časový odhad stanovený prideleným členom tímu. Ten, si svoju práci zaznamenáva priamo v nástroji JIRA. Priebeh šprintu je možné zobraziť pomocou burndown chartu, vygenerovaného priamo nástrojom JIRA.

6.1.5 Ukončenie šprintu

Po ukončení šprintu sa vypočíta súčet story pointov, za úspešne splnené úlohy. V prípade, že niektorá z úloh nebola počas šprintu dokončená, je vrátená do backlogu na prvé miesto. Táto úloha je následne vybratá na vykonanie v ďalšom šprinte.

6.2 Metodika pre verziovanie

Cieľom metodiky je sumarizácia pravidiel a odporúčaní pri verziovaní zdrojového kódu. Metodika je určená pre všetkých členov tímu.

6.2.1 Zaužívané pojmy

Verzovací systém (Git)

- systém umožňujúci verzovať súbory a z nich vytvárať vetvy. V projekte ide o systém Git[1].

Repozitár

- online úložisko zdrojového kódu kam majú všetci členovia tímu prístup
- využitie - synchronizácia / záloha / vytvorenie spustiteľných súborov (build) / nasadzovanie
- členovia tímu sem nahrávajú svoje úpravy zdrojových kódov
- náš repozitár - https://code.xit.camp/upvii

Vetva

- obsahuje množinu commit-ov - teda množinu zmien zdrojového kódu
- typy vetiev (podľa Gitflow):
    - master - špeciálna vetva - obsahuje kód, ktorý je vždy plne funkčný a nasaditeľný

[1] https://git-scm.com/

    - development - špeciálna vetva - obsahuje kód, ktorý je nasadzovaný na vývojový server, sp
    - feature vetvy - obsahuje zmeny sú súčasťou tej istej funkcionality
    - hotfix vetvy - obsahuje zmeny opravujúce chybu
- spájanie vetiev sa nazýva mergovanie

Commit

- zachytáva zmeny v zdrojovom kóde
- pomenovaný podľa konvencie (uvedená nižšie v metodike - sekcia 6.2.4 Vytváranie kontrolných bodov (commit-y))

6.2.2 Gitflow branching model

Gitflow predstavuje branching model zameraný na agilný spôsob vývoja. Tento model definuje rôzne typy vetiev, ktoré majú špecifické účely.

Hlavné vetvy:

- master vetva
    - obsahuje kód, ktorý je vždy plne funkčný, spĺňa definition of done a je okamžite nasaditeľný
- development vetva
    - kópia master vetvy
    - oproti master vetve obsahuje implementáciu, ktorá ešte nespĺňa definition of done, ale je potrebné ju otestovať na serveri
    - robí sa z nej deploy na vývojový server (staging server)
    - merge do master vetvy prebieha v čase vydania (release) - na konci šprintu

Vedľajšie vetvy:

- feature vetva
    - odvodzuje sa z development vetvy
    - používa sa pri vývoji novej funkcionality => vyvíja sa v nej nová funkcionalita (feature)
    - spojí sa development pomocou merge request-u
- hotfix vetva
    - odvodzuje sa z master vetvy
    - používa sa ak sa v produkčnej vetve (master) vyskytne chyba, ktorú treba opraviť
    - keď je chyba opravená, prebehne merge do master aj development vetvy

6.2.3 Vytváranie vetiev

Feature vetvy:

- sú odvodzované z development vetvy a pomenovávané nasledovne "*feature/[Jira-CisloTasku][kratky-popis-v-anglictine]*"

Hotfix vetvy:

- sú odvodzované z master vetvy a pomenovávané nasledovne "*hotfix/[Jira-CisloTasku][kratky-popis-v-anglictine]*"

Príklad pre task - "*IM-34 Vytvorenie ErrorHandlera a úprava súčasného chytania chýb*" by bol názov vetvy "*feature/IM-34-error-handling-refactor*"

Vytvorenie novej vetvy

"*git checkout -b* [NAZOV_NOVEJ_VETVY] [development / master]".

Príklad

"*git checkout -b feature/IM-34-error-handling-refactor development*".

6.2.4 Vytváranie kontrolných bodov (commit-y)

Commit:

- sa vytvára vždy keď je implementovaná ucelená časť funkcionality, pričom kód musí skompilovateľný
- je potrebné nahrať na repozitár (pushnúť)
- obsahuje deskriptívnu správu v angličtine - je nutné aby podľa nej bolo možné identifikovať zmeny, ktoré daný commit obsahuje (príklady commitov - https://chris.beams.io/posts/git-commit/#seven-rules)

Vytvorenie commitu:

- na vytvorenie môžeme použiť terminál alebo možnosti IDE. Uvádzam príklad pre terminál.
- " *git status* " - zobrazí zmenené súbory
- " *git add [FILE]* " - pridá súbor do vytváreného commitu
- " *git commit -m \"message\"* " - vytvorí commit
- " *git push origin [NAZOV VETVY]* " - nahrá zmeny na repozitár

V prípade potreby je možné na vytvorenie commitu použiť bez prepínača -m, kde môžeme zadať dlhšiu správu.

Po nahratí je na repozitári pustená kompilácia. Vždy treba skontrolovať, či prebehla korektne. Z vetiev development a master sa vytvára aj docker kontajner pre nasadenie na server.

6.2.5 Merge

Po implementovaní funkcionality vo feature (resp. hotfix) vetve sú tieto zmeny merge-nuté do development vetvy. Z development vetvy prebehne nasadenie na vývojový (staging) server. Po schválení produktovým vlastníkom (product owner, vedúci TP) sú zmeny mergnuté do master vetvy, odkiaľ sú nasadené na produkčný kód.

Poradie merge-ov:

- podľa gitflow - merge z feature vetiev prebieha vždy cez development!

Merge do developmentu

Merge môžem spraviť vždy pokiaľ verzia, ktorú chcem mergnúť:

- skompilovateľná
- funkčná stará funkcionalita
- nová funkcionalita nesmie vyhadzovat chyby
- v novej verzii nebolo zmenené API - pokiaľ bolo zmenené môžem mergnúť len ak bude upravená aj development vetva opačnej strany (frontend - backend)

Merge do master

- len z developmentu (prípadne hotfix)
- ak bol development nasadený a otestovaný na vývojovom serveri
- ak bola verzia na vývojovom serveri akceptovaná product ownerom (vedúcim)

Postup pri merge:

- vytvorenie merge request-u - najjednoduchšie cez webové rozhranie
- prebehne prehliadka zdrojového kódu (code review), ktorá sa riadi metodikou pre prehliadky zdrojového kódu
- po zapracovaní zmien vývojárom je možné spraviť merge
    - väčšinou možné cez webové rozhranie - neodporúčané
    - lokálne, postup je uvedený vo webovom rozhraní gitlab-u. Odporúča sa po mergi minimálne skompilovať program.

6.2.6 Nasadenie

Zo zdrojových kódov vo vetvách development a master sa po kompilácii vytvoria docker kontajnery, ktoré sa nasadzujú na vývojový, resp. produkčný server. Postup nasadzovania je uvedený v metodike pre nasadzovanie.
vetva development -> staging server
vetva master -> production server

6.3 Metodika pre nasadzovanie

6.3.1 Zaužívané pojmy

Docker

- sú nástroje na virtualizáciu, pričom viaceré virtuálne stroje zdieľajú jedno jadro OS
- umožňuje jednoduchšie nasadzovanie
- jednotlivé programy pracujú v tzv. kontajneroch, ktoré medzi sebou dokážu komunikovať

Docker-compose

- nástroj na manažovanie docker kontajnerov
- umožňuje spustenie viacerých kontajnerov spolu s ich inicializáciou

Elasticsearch

- distribuovateľný, škálovateľný vyhľadávací engine
- používa sa napríklad na autocomplete

PostgreSql

- databázový server

6.3.2 Kedy sa nasadzuje

Produkčný server (Production):

- keď sa vydáva nová verzia - t.j. na konci šprintu z vetvy master

Vývojový server (Staging):

- priebežne počas šprintu - najskôr ako je to možné (po merge do developmentu)

6.3.3 Inicializácia servera

Server

- používaný server je od Google - gcloud
- návod na inicializáciu je dostupný na wiki

Technológie

- používaný operačný systém - CentOS
- nainštalovať docker, docker-compose - podrobný návod dostupný na wiki
- docker-compose slúži na spustenie všetkých potrebných služieb:

    - Elasticsearch
    - PostgreSql
    - PgAdmin
    - service (backend)
    - web (frontend)
- *docker-compose.yml* - konfiguračné súbory pre docker-compose sa nachádzajú v repozitári *docs* na Gitlabe.
    - pre príslušný server (staging / production) je potrebné skopírovať *docker-compose.yml* na server

Inicializácia Elasticsearch (podrobný návod na wiki)
- Lemmagen plugin + slovenčina (potom reštart elasticsearchu)
- vytvorenie indexov (súbor *createIndex.sh* v repozitári *docs*)
- nahodenie centrálneho modelu do indexov (*ElasticIndex.java*)

Inicializácia PostgreSQL
- je potrebné vytvoriť databázu "*document*" s používateľom "*postgres*" a heslom "*postgres*"
- o vytvorenie tabuliek sa postará framework

6.3.4 Nasadenie

Samotné nasadenie prebieha po zostavení riešenia v *deploy* časti automatického build-u na repozitári Gitlab za predpokladu, že frontend a backend sú navzájom kompatibilné (API).
- po zostavení riešenia sa vytvorí kontajner pre docker a uloží sa na úložisko
- stiahnutia a spustenie nového riešenia ("*docker-compose up -d service*" (prípadne *web* namiesto *services*)
- podrobný postup dostupný na wiki

6.4 Metodika pre vývoj a prehliadku zdrojového kódu

6.4.1 Postup

Všetok napísaný kód súvisí s úlohami, ktoré sú vytvorené v nástroji JIRA. Životný cyklus týchto úloh je popísaný v metodike manažmentu úloh. Tento dokument bližšie rozoberá stavy Ready to Review, In Review a Ready to Deploy.

1. Keď je programátor, ktorý pracoval na úlohe, spokojný s vykonanou prácou a skontroloval jej funkčnosť, presunie svoju úlohu do stavu Ready to Review. Zároveň s týmto presunom musí programátor v nástroji Gitlab otvoriť Merge Request - požiadavku na merge do vetvy development.

2. Kontrolovač si stiahne zdrojový kód z vetvy, ktorú programátor použil na vývoj danej funkcionality.
3. Kontrolovač krátko skontroluje, či dodaný kód funguje, ako má.
4. Následne robí prehliadku kódu na základe pravidiel písania zdrojového kódu, ktoré sú uvedené nižšie.
5. V prípade, že kontrolovač nájde niečo, čo nevyhovuje popísanej kvalite, pomocou nástroja Gitlab nechá autorovi Merge Requestu komentár, na ktorý musí programátor reagovať. Výsledkom je diskusia, ktorá končí jednou z dvoch možností:
    a. V prípade, že programátor má pádne argumenty vysvetľujúce a zdôvodňujúce diskutovaný kód, kontrolovač uzavrie diskusiu bez ďalších zmien.
    b. Ak je kód programátora naozaj chybný, musí programátor pripraviť nový commit, ktorý adresuje pripomienky kontrolovača. Ten následne tento commit zhodnotí opäť - výsledkom je buď ďalšie pokračovanie diskusie, alebo jej uzavretie.
6. Ak sú všetky diskusie uzavreté, kontrolovač urobí jednu z nasledujúcich akcií:
    a. Ak kód vyhovuje štandardom kvality, vetva s kódom sa mergne do vetvy development podľa metodiky pre verziovanie.
    b. Ak kód nevyhovuje štandardom kvality, Merge Request je zamietnutý bez vykonania mergu.

6.4.2 Pravidlá písania zdrojového kódu

Vyvíjaný projekt je aktuálne rozdelený na 2 časti:

- Frontend písaný v jazyku TypeScript s využitím rámca Angular
- Backend písaný v jazyku Java s využitím rámca Spring Boot

Táto sekcia je v dôsledku tejto skutočnosti tiež rozdelená na 2 časti - každá rozoberá jednu časť projektu.

Prvé pravidlo, ktoré je dôležitejšie než všetky ostatné, je: **Používať common sense**. Ak niečo dáva zmysel, funguje to a je to elegantné, má to prioritu pred týmto dokumentom.

TypeScript - Angular

Používať pravidlá definované súborom tslint.json - množstvo nájstrojov poskytuje zvýrazňovanie chýb na základe pravidiel definovaných v tomto súbore.

Typy

Treba využívať typy čo najviac, ako sa len dá. Slabinou jazyka TypeScript je, že predstavuje nadstavbu nad JavaScriptom, takže je jednoduché zabudnúť uviesť typ - treba si na to dávať pozor. Kód bez typov je síce kratší, ale nezachytí takmer žiadne chyby.

Zle (posielanie HTTP požiadaviek):

```
public saveAttribute(model, token) {
    const result = {
        uri: '',
        label: model.label,
        owlCategory: model.owlCategory,
        group: '',
        description: model.description,
        similarPhrase: Array.isArray(model.similarPhrase) ? model.similarPhrase :
[model.similarPhrase],
        flag: true,
    };

    return this.http.post(`${AppConfig.SERVER}/addAttribute`, result, {
        headers: this.getHeaders(),
        params: { token },
    });
}
```

Dobre (posielanie HTTP požiadaviek):

```
public saveAttribute(model: OWLDataObject, token: string): Observable<OWLDataObject> {
    const result: OWLDataObject = {
        uri: '',
        label: model.label,
        owlCategory: model.owlCategory,
        group: '',
        description: model.description,
        similarPhrase: Array.isArray(model.similarPhrase) ? model.similarPhrase :
[model.similarPhrase],
        flag: true,
    };

    return this.http.post<OWLDataObject>(`${AppConfig.SERVER}/addAttribute`, result, {
        headers: this.getHeaders(),
        params: { token },
    });
}
```

Zle (prijímanie HTTP požiadaviek):

```
this.findAttributeService.find(this.searchQuery).subscribe(
    (values) => {
```

```
        this.possibilities = values;
        this.paginator.pages = Math.ceil(values.length / this.paginator.perPage);
    },
    (error) => {
        this.notification.showErrorTranslatedHttp(
            'attributeSearch.error.search.title',
            'attributeSearch.error.search.message',
            error
        );
    },
);
```

Dobre (prijímanie HTTP požiadaviek):

```
this.findAttributeService.find(this.searchQuery).subscribe(
    (values: OWLDataObject[]) => {
        this.possibilities = values;
        this.paginator.pages = Math.ceil(values.length / this.paginator.perPage);
    },
    (error: HttpErrorResponse) => {
        this.notification.showErrorTranslatedHttp(
            'attributeSearch.error.search.title',
            'attributeSearch.error.search.message',
            error
        );
    },
);
```

Organizácia súborov

Všetky súbory súvisiace s komponentom by mali ísť spolu do jednej zložky (služby (service), testy, pomocné triedy...).

Oddelene (v zdieľanej zložke) by sa mali nachádzať len:

- naozaj zdieľané komponenty (autocomplete, dropdown, spinner...)
- model pre sieťovú komunikáciu s backendom

HTTP komunikácia (a chybové stavy všeobecne)

Vždy je potrebné ošetriť chybový stav a zobraziť používateľovi informáciu o tom, že sa niečo pokazilo (s čo najpresnejším popisom).

Zle:

```
this.findAttributeService.find(this.searchQuery).subscribe(
   (values: OWLDataObject[]) => {
      this.possibilities = values;
      this.paginator.pages = Math.ceil(values.length / this.paginator.perPage);
   },
);
```

Dobre:

```
this.findAttributeService.find(this.searchQuery).subscribe(
   (values: OWLDataObject[]) => {
      this.possibilities = values;
      this.paginator.pages = Math.ceil(values.length / this.paginator.perPage);
   },
   (error: HttpErrorResponse) => {
      this.notification.showErrorTranslatedHttp(
         'attributeSearch.error.search.title',
         'attributeSearch.error.search.message',
         error
      );
   },
);
```

Nemanipulovať s DOM priamo (jQuery a podobne)

V starom frontende (pred refaktorom) bolo na veľa miestach používané jQuery na ručnú manipuláciu s DOM. Angular bol stvorený práve kvôli tomu, aby toto už nebolo potrebné. Na strane TypeScriptu sa musí riešiť logika, nie dizajn!

Zle (zo starého frontendu):

```
   // Vyskytuje sa tu veľa logiky súvisiacej s rozložením stránky.
   // Nič z toho nie je potrebné!
   // Vždy je potrebné riešiť dizajn pomocou HTML a SCSS, do TypeScriptu
   // treba zasahovať len v prípade krajnej núdze (tento príklad v krajnej
   // núdzi nebol, po vymazaní dizajnovej logiky z TypeScriptu sa stránka
   // dokonca zobrazovala lepšie).

   public moveTable(value: number): void {
      this.maxOffset = $("#tableHead").width() - $(".pane-hScroll").width();
      this.actualOffset += value * EditComponent.SPEED;
      this.actualOffset = Math.max(0, this.actualOffset);
```

```
        $(".pane-hScroll").scrollLeft(this.actualOffset);
    }

    public ngAfterViewInit(): void {
        this.actualOffset = 0;
        this.maxOffset = $("#tableHead").width() - $(".pane-hScroll").width();
        $("th").each(function(index) {
            const th = $(this);
            const width = th.find(".d-inline-block").width();
            $("td:nth-child(" + (index + 1) + ")").width(width);
            th.width(width);
        });
        this.cdr.detectChanges();
    }

    public ngOnInit(): void {
        this.entityService.getEntities().subscribe(
            (response: Entity[]) => this.entities = response,
            (error) => this.notificationService.showErrorMessage(error));

        // this.entities = this.entityService.fake_getEntities();

        this.maxOffset = $("#tableHead").width() - $(".pane-hScroll").width();
        this.cdr.detectChanges();

            ...
    }
```

Dobre (čo zo starého kódu zostalo):

```
    public ngOnInit(): void {
        this.entityService.getEntities().subscribe(
            (response: ResponseObject<EntitiesContent>) => this.entities =
response.content.entities,
            (error: HttpErrorResponse) => {
                this.notification.showErrorTranslatedHttp('edit.error.init.title', 'edit.error.init.message',
error);
            },
        );
    }
```

Využívať niektoré princípy funkcionálneho programovania (map, filter...)

Možno trochu kontroverznejší bod metodiky, ale najmä v situáciách, kedy sa zoznam jedného typu transformuje na iný zoznam obsahujúci iný typ hodnôt sa kód väčšinou zjednoduší pri použití metódy map (v kombinácii s inými funkciami, napríklad filter - článok pre inšpiráciu: https://medium.com/poka-techblog/simplify-your-javascript-use-map-reduce-and-filter-bd02c593cc2d)

Zle:

```
this.columnData = []
for (const row of this.fileData.uniqueData) {
    const el = row[this.columnIndex];
    if (el !== null) {
        this.columnData.push(el);
    }
}
```

Dobre:

```
this.columnData = this.fileData.uniqueData
    .map((row) => row[this.columnIndex])
    .filter((el) => el !== null);
```

Java - Spring Boot

Formát

Všetok kód je formátovaný defaultným formatterom v IntelliJ. Toto formátovanie sa dá spustiť klávesovou skratkou Ctrl + Shift + L.

Try catch

Vo väčšine prípadov treba pri ošetrení výnimky preferovať deklaráciu throws
pred try - catch blokom.

Zle:

```
public String doStuff(String path) {
    try {
        return readFile(path);
    } catch (IOException e) {
        logger.error(e);
```

```
    }
}
```

Dobre:

```
public String doStuff(String path) throws IOException {
    return readFile(path);
}
```

Ak sa jedná o chybu, ktorá je očakávaná a program po nej môže pokračovať, môže byť vhodnejšie použiť aj try - catch:

```
public int getWithDefault(String key, int defaultValue) {
    try {
        return storage.getValue(key);
    } catch (KeyNotFoundException e) {
        return defaultValue;
    }
}
```

Uzatváranie zdrojov (resource) a try-catch with resources

Ak zatvárame nejaký zdroj dát (napr. File), treba použiť
try-catch with resources:

Zle:

```
public void saveFile(long id) throws IOException {
    File file = openFile(id);
    processFile(file);
    file.close();
}
```

Dobre:

```
public void saveFile(long id) throws IOException {
    try (File file = openFile(id)) {
        processFile(file);
    }
}
```

Životnosť premenných

Všetky premenné by mali mať najkratšiu možnú životnosť a mali by byť deklarované čo najbližšie ich použitiu:

Zle:

```
public ResponseObject getResponseObject() {
   ResponseObject responseObject = new ResponseObject();

   FileContent fileContent = new FileContent();
   fileContent.setData(fileService.getData());

   responseObject.setContent(fileContent);
   return responseObject;
}
```

Dobre:

```
public ResponseObject getResponseObject() {
   FileContent fileContent = new FileContent();
   fileContent.setData(fileService.getData());

   ResponseObject responseObject = new ResponseObject();
   responseObject.setContent(fileContent);
   return responseObject;
}
```

Zbytočný stav

Tiež je preferované nezneužívať členské premenné na účel posúvania argumentov. Členské premenné sú potrebné na uchovávanie stavu objektu - ak nie je uchovávanie stavu potrebné, je lepšie sa mu vyhnúť.

Zle:

```
public class FileReader {
   private String path;

   public FileReader(String path) {
      this.path = path;
   }

   public byte[] read() {
      File file = new File(path);
      return IOUtil.readBytes(file);
   }
}
public class Main {
```

```
    public static void main(String[] args) {
        FileReader reader = new FileReader("/path/to/file.txt");
        byte[] data = reader.read();
        ...
    }
}
```

Lepšie:

```
public class FileReader {
    public static byte[] read(String path) {
        File file = new File(path);
        return IOUtil.readBytes(file);
    }
}
public class Main {
    public static void main(String[] args) {
        byte[] data = FileReader.read("/path/to/file.txt");
        ...
    }
}
```

Spring Boot spôsob:

Tento spôsob je preferovaný.

```
@Service
public class FileReader {
    public byte[] read(String path) {
        File file = new File(path);
        return IOUtil.readBytes(file);
    }
}
@Controller
public class SomeController {
    @Autowired
    private FileReader fileReader;

    public void someFunction() {
        byte[] data = fileReader.read("/path/to/file.txt");
        ...
    }
}
```

6.5 Metodika pre komunikáciu

Cieľom metodiky je sumarizácia pravidiel a odporúčaní pri oficiálnej komunikácií vrámci tímu pre záležitosti týkajúce sa tímového projektu. Metodika je určená pre všetkých členov tímu.

6.5.1 Zaužívané pojmy

Slack

- systém na sústredenie väčšiny písanej tímovej komunikácie
- funguje na princípe kanálov, do ktorých môžu členovia tímu napísať svoje príspevky, na základe povahy samotného kánalu a správy
- tímový Slack sa nachádza na https://team16-fiit-tp-18-19.slack.com/

Standup

- na základe princípov agilného vývoja, tím organizuje v pravidelných intervaloch standupy
- standup sa koná po celej dohode tímu 4x do týždňa (v nedeľu večer, utorok večer, štvrtok večer a piatok na úvode stretnutia s product ownerom okolo poludnia)
- standupy v nedeľu a utorok sa môžu konať online, na Slacku, po dohode tímu
- ich cieľom je vzájomné informovanie o progrese, identifikovaných problémoch a práci na vybraných úlohách

Zápisnica a záznamy stretnutí

- z každého oficiálneho stretnutia s product ownerom / vedúcim tímu sa vedie zápisnica, pričom by mala byť zhotovená aj z tímových stretnutí v štvrtky večer
- stretnutia s vedúcim tímu budú nahrávané, aby sme následne vedeli doplniť zápisnicu o prípadné vynechané postrehy
- zápisnica slúži ako prehľad prediskutovaných tém, poznámok a sumár kľúčových informácií odovzdaných počas stretnutia a je braná ako forma komunikácie tímu “samého so sebou”, kedy tím môže vyhľadať informáciu z “minulosti” na ktorú už diskutoval a potrebuje sa dozvedieť detaily

6.5.2 Organizácia kanálov Slacku

Na Slacku majú všetci členovia tímu prístup do vybraných kanálov. Okrem členov tímu má doň prístup vedúci tímu / product owner, ďalej Marek Šurek a Matúš Brandajský, aby sme mohli komunikovať aj s nimi, najmä na začiatku projektu v prvých týždňoch.
Organizácia kanálov na Slacku je nasledovná:

Cloud

Kanál slúži na informovanie o stave serverov a sú v ňom uložené aj kľúčové informácie o nich, napr. IP adresy.

General

Všeobecný kanál, ktorý slúži na komunikáciu so všetkými členmi tímu, ale nie je odporúčaný. Používa sa, iba ak správa nevyhovuje umiestneniu na ostatných kanáloch, no je dôležité, aby o nej vedel celý tím.

Jira

Kanál cez ktorý bola integrovaná tímová Jira so Slackom, posielajú sa doňho informácie o zmenách v produktovom backlogu či aktuálnom šprinte, no môžu doň prispievať aj členovia tímu ohľadom vecí týkajúcich sa scrum boardu či backlogu.

Mts

Kanál pre 4 členov absolvujúcich predmet MTS, vrámci ktorého boli riešené finálne úpravy metodík a dokumentácie a príprava prezentácie.

Random

Všeobecný kanál pre menej dôležité informácie týkajúce sa tímového projektu.

Standupy

Kanál na realizáciu online standupov, patria sem členovia tímu.

Stretnutia

Kanál na dohadovanie termínov stretnutí a ich organizácie.

Team_private_channel

Kanál používaný výlučne členmi tímu na väčšinu všeobecnej komunikácie, ktorou tím nechce zaťažovať vedúceho tímu / product ownera a ostatných.

Na-pivo

Kanál na organizácie networkingu. Patria sem členovia tímu a vedúci tímu. Využívaný zatiaľ žiaľ minimálne.

Priame používateľské kanály

Slúžia na komunikáciu s konkrétnym členom tímu, ak sa povaha komunikácie týka len dotyčnej osoby.

6.5.3 Prispievanie do Slacku

Pri prispievaní na Slack sa všetci používatelia riadia tým, že volia správny kanál na odkomunikovanie informácie. Spravidla je kanál Random a kanál General použitý len v prípade, že iné kanály nevyhovujú, teda je informácia buď príliš všeobecná alebo príliš konkrétna.

V rámci oficiálnych kanálov slúžiacich na tímové záležitosti, teda Jira, General, Standupy, Stretnutia, je členovia tímu berú na vedomie, že informácie na nich môžu byť viditeľné pre vedúcich tímu alebo môžu slúžiť aj na podklad pre zápisnice (napr. Standupy) či dokumentáciu, preto sa na nich vyjadrujú vecne a vhodne.

6.5.4 Realizácia standupov

Standupy sa organizujú po dohode celého tímu (osobne alebo na Slacku -> Standupy) v stanovené dni (Nedeľa večer, Utorok večer, Štvrtok večer, Piatok stretnutie s vedúcim okolo poludnia).
Standupy by mali byť krátke, vecné, stručné, odporúčaná doba hovorenia na každého člena je 30 sekúnd až 1 minúta. Scrum master riadi standupy, takže pri zhromaždení tímu udelí každému slovo, prípadne urýchľuje či zastavuje ľudí hovoriacich pridlho, príliš podrobne alebo od veci, ďalej môže uviesť krátku sumarizáciu na záver a následne rozpustí standup.
Každý člen tímu výstižne a stručne zhrnie počas svojej časti, čo od posledného standupu urobil na tímový projekt, s čím pokročil, na aké problémy narazil a čo má ešte na pláne. V dôsledku iných školských povinností je v poriadku, ak niekto na standupe vyhlási, že nemal čas sa momentálne venovať tímovému projektu, hoci by sa to nemalo opakovať dvakrát po sebe, čo by predznačovalo potenciálny problém pri plnení tímových úloh.
Ak sa tím stretáva na Slacku na kanáli Standupy, tak v zvolenom čase uvedie každý svoj príspevok zahŕňajúci to, čo by povedal na fyzickom standupe. Takéto standupy nie sú riadené scrum masterom.
V záujme urýchlenia fyzických standupov môže tím využiť špeciálnu verziu standupu, tzv. plankup. Členova tímu sa položia na zem do horizontálnej polohy tvárou nadol a zdvihnú váhu svojho tela opierajúc sa o predlaktia, vystrú chrbát a boky (poloha zvaná plank) a vedú standup “plankup” v tejto polohe. Priemerný fyzicky zdatný človek by mal udržať túto polohu 2 minúty, čo by však malo vždy byť postačujúce na odkomunikovanie kľúčových informácií medzi 6 ľuďmi.

6.5.5 Realizácia stretnutí

Vrámci každého stretnutia je Scrum master (aktuálny) zodpovedný za agendu a vedenie stretnutia. Pri stretnutiach s vedúcim tímu / product ownerom, môže samozrejme prebrať iniciatívu on, no spravidla, Scrum master by mal byť pripravený na rolu vedenia stretnutí.

Riadi aj toky komunikácie počas nich, teda udeľuje slovo, prípadne môže prerušiť ľudí, ktorí neprispievajú vecne do komunikácie napríklad počas standupov. Riadi prezentáciu produktu na konci šprintu, alebo poveruje riadením iného člena tímu, vedie retrospektívu a je zodpovedný za poverenie člena tímu písaním zápisnice. Je zodpovedný aj za organizáciu priebeh standupov.

6.5.6 Vedenie zápisníc a záznamov zo stretnutí

Na každom stretnutí aspoň jeden člen tímu nahráva záznam stretnutia na mobil alebo počítač a aspoň jeden člen tímu (pokojne ten istý) vykonáva zápis.
Zapisuje sa priebežne všetko kľúčové, čo sa preberá, pričom po ukončení stretnutia sa môže zápisnica prejsť, upraviť na prehľadnejší tvar. Jej štruktúra nie je striktne definovaná, jedná sa predovšetkým len o typ záznamu, ktorý má pomôcť tímu pri spätnom vyhľadaní informácií. Má byť stručná, jasná, heslovitá, môže byť písaná v odrážkach.
Záznamy sa môžu uchovať, no ich obsah by sa mal finálne pretransformovať do dodatočných informácií v zápisnici, aby si ich členovia tímu nemuseli prehrávať.
Zápisnice sa zverejňujú po stretnutiach na tímovom webe a wiki.

6.6 Metodika práce

Okrem vyššie uvedeného z doterajších retrospektív vyplynulo:

- v polovici šprintu mať začaté tasky
- organizovať standupy každý druhý deň. Ak sa nedá osobne, tak cez Slack.
- v Jire logovať prácu na taskoch (hodiny)
- dodržiavať metodiky - Gitlab, štýl programovania
- pokiaľ to dáva zmysel písať automatické testy
- zapájať viac ľudí do code review
- obmieňať pracovné oblasti (šírenie vedomostí)
- ak to dáva zmysel, využívať techniku párového programovania
- ak je čas, pridať do šprintu tasky za 0 story points (epic Infraštruktúra)
- v priebehu práce na implementačnom tasku pridať malý refaktor (pokiaľ sa stíha)
- pokračovať v promptných code review

7. Webové sídlo projektu

Webové sídlo projektu je dostupné na adrese:
http://labss2.fiit.stuba.sk/TeamProject/2018/team16iss-it/.
Obsahuje zoznam členov tímu, základné informácie o projekte a v časti ‘Dokumentácia’ zoznam šprintov s ich krátkych opisom. Jednotlivé šprinty obsahujú odkazy na zápisnice zo stretnutí tímu. Okrem toho je na webovom sídle projektu odkaz na wiki projektu. Pre prístup na ňu je potrebné heslo.