Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Projektová dokumentácia

(Tím 01 – sUXess)

Akademický rok: 2015/2016
Predmet: Tímový projekt

Študenti: Bc. Dubec Peter
Bc. Róbert Cuprik
Bc. Gajdošík Patrik
Bc. Roba Roman
Bc. Sanyová Monika
Bc. Vrba Jakub
Bc. Žigo Tomáš

Vedúci tímu: Ing. Móro Róbert

Obsah

1 Riadenie projektu **4**
1.1 Úvod . . . 4
1.2 Roly členov tímu a podiel práce . . . 4
1.2.1 Roly členov tímu . . . 4
1.2.2 Podiel na dokumentácii riadenia projektu . . . 5
1.2.3 Podiel na dokumentácii k inžinierskemu dielu . . . 5
1.3 Aplikácie manažmentov . . . 6
1.3.1 Procesy manažmentu projektu . . . 6
1.3.2 Procesy manažmentu softvéru . . . 7
1.3.3 Procesy vývoja a prevádzky softvéru . . . 7
1.4 Sumarizácie šprintov . . . 8
1.4.1 1. šprint . . . 8
1.4.2 2. šprint . . . 9
1.4.3 3. šprint . . . 10
1.4.4 4. šprint . . . 11
1.4.5 5. šprint . . . 12
1.4.6 6. šprint . . . 12
1.4.7 7. šprint . . . 13
1.4.8 8. šprint . . . 14
1.4.9 9. šprint . . . 14
1.4.10 10. šprint . . . 15
1.4.11 11. šprint . . . 15
1.4.12 Vyhodonotenie šprintov . . . 16
1.5 Používané metodiky . . . 17
1.5.1 Metodika verziovania . . . 17
1.5.2 Metodika revízie kódu . . . 17
1.5.3 Metodika kontinuálnej integrácie . . . 17
1.5.4 Metodika pre správu riadenia projektu a požiadaviek . . . 18
1.5.5 Metodika pre písanie kódu v JavaScripte . . . 18
1.5.6 Metodika pre písanie kódu v Ruby on Rails . . . 18
1.5.7 Metodiky pre testovanie aplikácie . . . 18
1.6 Globálna retrospektíva . . . 18

2 Inžinierske dielo **21**
2.1 Úvod . . . 21
2.2 Ciele projektu . . . 21
2.2.1 Zadefinovaný MVP . . . 22
2.2.2 Vytvorený prototyp . . . 23
2.3 Architektúra . . . 25
2.3.1 Frontend . . . 26
2.3.2 Backend . . . 27
2.3.3 Komponenty systému . . . 28
2.3.4 Dátový model . . . 29
2.3.5 Tímový server . . . 31
2.4 Moduly systému . . . 32
2.4.1 Nahrávanie obrázkov . . . 32

2.4.2 Mockup – Editor 32
2.4.3 Zaznamenávanie aktivity používateľov 34
2.4.4 Vyhodnocovanie testov a práca so záznamami 35

A Zoznam kompetencií tímu **37**

B Metodiky **39**

C Export evidencie úloh **62**

D Inštalačná príručka **75**
D.1 Inštalácia a spustenie projektu 75
D.2 Príprava vývojárskeho prostredia 75
D.3 Nasadenie do produkčného prostredia 76

E Používateľská príručka **77**
E.1 Správa projektov 77
E.2 Správa prototypov 79
E.3 Správa testov 79
E.3.1 Tasky 81
E.3.2 Priebeh testu 81
E.3.3 Náhľad testu 82
E.4 Výsledky testov 83

F Používateľská príručka pre editor **89**

1 Riadenie projektu

1.1 Úvod

Pri práci na projekte v tíme je potrebné stanoviť procesy, ktoré v tíme prebiehajú, rovnako ako aj spôsoby a kritéria, akými tieto procesy majú prebiehať.

Tento dokument predstavuje dokumentáciu ku riadeniu nášho tímového projektu. Obsahuje opis postupov a metód, ktorými sme sa počas procesu práce na projekte riadili. V časti 1.2 sú opísané role, ktoré prislúchali jednotlivým členom, oblasti, za ktoré boli zodpovední a úlohy, ktoré vrámci tímu vykonávali. Nachádza sa tam takisto vyhodnotenie toho, akou časťou sa členovia tímu podieľali na tvorbe diela, ako aj ich podiel práce na tomto dokumente a na dokumentácií k inžinierskemu dielu.

Nasleduje časť 1.3, kde je opísané a vyhodnotené plnenie jednotlivých manažérskych úloh členmi, ktorí za ne boli zodpovední. Keďže sme sa v našom tíme riadili agilnou technikou Scrum, v časti 1.4 sú zosumarizované šprinty, ktoré prebehli. Dôležitou časťou riadenia projektu v tíme sú metodiky, ktoré určujú pravidlá, ktorými je v tíme potrebné sa riadiť. Metodiky je možné nájsť v časti 1.5. Časť 1.6 potom následne obsahuje retrospektívu k práci v tíme rozdelené podľa semestrov.

1.2 Roly členov tímu a podiel práce

1.2.1 Roly členov tímu

Každý člen tímu má pridelenú určitú oblasť, za ktorú je zodpovedný alebo na ktorej pracuje:

Jakub Vrba (manažér dokumentácie):

- práca na frontend-e (AngularJS) – vývoj a testovanie

Monika Sanyová (manažérka testovania):

- práca na backend-e (RoR) – vývoj a testovanie
- návrh API

Patrik Gajdošík (manažér testovania):

- kontinuálna integrácia
- prevádzka tímového servera

Peter Dubec (vedúci tímu, manažér verzií):

- návrh API
- návrh používateľského prostredia pre frontend

Roman Roba (manažér riadenia v tíme):

- JIRA

Róbert Cuprik (manažér kvality):

- Code review
- autentifikácia používateľa v API
- práca na frontende - vývoj a testovanie

Tomáš Žigo (manažér zdrojov):

- frontend - vývoj a návrh používateľského prostredia

Kapitola	Vypracoval
1.1 Úvod	Patrik Gajdošík
1.2 Roly členov tímu a podiel práce	Patrik Gajdošík, Jakub Vrba
1.3 Aplikácie manažmentov	Tomáš Žigo
1.4 Sumarizácie šprintov	Roman Roba
1.5 Používané metodiky	Jakub Vrba
1.6 Globálna retrospektíva	Roman Roba, Peter Dubec

Tabuľka 1: Podiel na dokumentácii riadenia projektu

Kapitola	Vypracoval
2.1 Úvod	Roman Roba
2.2 Ciele projektu	Peter Dubec
2.3 Architektúra	Tomáš Žigo
2.3.1 Frontend	Róbert Cuprik, Jakub Vrba
2.3.2 Backend	Peter Dubec
2.3.3 Komponenty systému	Róbert Cuprik
2.3.4 Dátový model	Monika Sanyová
2.3.5 Tímový server	Patrik Gajdošík
2.4.1 Nahrávanie obrázkov	Patrik Gajdošík
2.4.2 Mockup – Editor	Patrik Gajdošík, Róbert Cuprik

Tabuľka 2: Podiel na dokumentácii inžinierskeho diela

1.2.2 Podiel na dokumentácii riadenia projektu

Podiel práce členov tímov na dokumentácii k riadeniu projektu je možné vidieť v tabuľke 1.

1.2.3 Podiel na dokumentácii k inžinierskemu dielu

Podiel práce členov tímov na dokumentácii k inžinierskemu dielu je možné vidieť v tabuľke 2.

1.3 Aplikácie manažmentov

1.3.1 Procesy manažmentu projektu

Manažment komunikácie:

- **organizácia komunikácie v tíme** – Komunikácia v tíme je organizovaná na dve časti: tímové stretnutia a komunikácia mimo tímových stretnutí, na ktorú využívame nástroj na tímovú komunikáciu Slack, ktorý je voľne dostupný. Pomocou tohto nástroja je možné kedykoľvek kontaktovať konkrétneho človeka, prípadne skupinu ľudí, ktorým potrebujem niečo oznámiť alebo sa ich na niečo spýtať. Na tímových stretnutiach prebieha komunikácia tak, že ak má niekto nejaký návrh alebo pripomienku, prednesiu ju pred zvyškom tímu a následne prebehne diskusia.
- **informovanie vedúceho o stave projektu** – Produktový vlastník, v našom prípade je to vedúci tímu, je informovaný o stave projektu na každom tímovom stretnutí, ktoré sa koná každý týždeň. Na stretnutí každý člen tímu zrekapituluje dosiahnutý pokrok za uplynulý týždeň a na konci šprintu je zrekapitulovaný aj aktuálny stav projektu.

Manažment rozvrhu (časové plánovanie):

- **plánovanie pre tím a jednotlivých členov tímu** – Vývoj softvéru je rozdeľovaný do šprintov, kde každý šprint trvá dva týždne. V strede každého šprintu – po prvom týždni – sa koná priebežné stretnutie, na ktorom je diskutovaný aktuálne dosiahnutý pokrok a prípadné problémy, ktoré mohli nastať. Na začiatku, resp. konci, každého šprintu sú naplánované úlohy, ktoré by sa mali spraviť v priebehu aktuálneho šprintu a tieto jednotlivé úlohy sú pridelené konkrétnemu členovi tímu, ktorý sa tak stáva zodpovedným za splnenie tejto úlohy.
- **udržiavanie informácií o stave projektu** – Stav projektu je reprezentovaný úlohami, ktoré sú aktuálne riešené, budú riešené a boli už vyriešené. Všetky úlohy sú evidované nástrojom Jira na správu riadenia projektu a požiadaviek. V nástroji Jira si môže každý člen tímu kedykoľvek pozrieť úlohy, ktoré mu boli pridelené v aktuálnom šprinte, prípadne aj všetky ostatné úlohy, ktoré projekt obsahuje.
- **evidencia úloh** – V nástroji na správu riadenia projektu a požiadaviek Jira je možné prehliadať všetky prideľené úlohy, ich ohodnotenie, krátky popis, čo úloha predstavuje a v akom stave riešenia / splnenia sa nachádza.
- **vyhodnocovanie plnenia plánu a návrh úprav** – Na konci šprintu – dva týždne po jeho začiatku – prebehne na tímovom stretnutí krátka prezentácia splnených úloh, uzavretie úloh produktovým vlastníkom ak sú splnené korektne a retrospektíva ukončeného šprintu, v ktorej sa každý člen vyjadrí k tomu, čo podľa neho prebiehalo v poriadku, čo by sa malo začať robiť, aby šprinty prebiehali lepšie a čo by sa malo prestať robiť.

Manažment rozsahu (manažment úloh):

- **stanovenie sledovaných charakteristík produktu** – Na začiatku vývoja softvéru boli predstavené všetky návrhy na rôzne funkcie, ktoré by mohol finálny produkt obsahovať. Na tímovom stretnutí prebehol brainstorming, z ktorého tieto nápady vyšli. Následne prebehla diskusia, na ktorej produktový vlastník definoval požiadavky na minimálnu funcionalitu produktu. Z minimálnej funkcionality vyšli základné úlohy, ktoré boli porozdeľované do menších podúloh a ohodnotené.

Manažment kvality (manažment zmien, manažment chýb):

- **monitorovanie, prehliadky vytváraného výsledku (code review, testy)** – Kvalita jednotlivých funkcií, resp. úloh, produktu je zabezpečovaná samotným členom, ktorý je zodpovedný za ich splnenie.

Po skončení implementácie každý člen vykoná „code review“, čo je revízia svojho kódu, aby kód bol syntakticky správny a dobre čitateľný. Pod dobre čitateľným kódom sa rozumie forma kódu, ktorá spĺňa vopred stanovené normy. Okrem revízie kódu členom zodpovedným za danú úlohu je kontrola vykonávaná aj ostatnými členmi tímu pri dokončení úlohy a následne vytvorenom pull requeste alebo pri prezentácii dokončenej úlohy na tímovom stretnutí. Ako pomôcku pre revíziu kódu na frontende, ktorý je písaný v jazyku Javascript používame tiež nástroj Jshint. Kontrola týmto nástrojom je vykonávaná aj v kontinuálnej integrácii, čím zabezpečujeme, že v produkcii bude vždy zrevidovaný kód. Bližšie informácie o tom ako vykonávať revíziu svojho kódu a ako má výsledný kód vyzerať sú uvedené v metodike revízie kódu.

Každú funkciu je tiež potrebné otestovať, aby bola zaistená potrebná kvalita tejto funkcie. Rovnako informácie o tom ako testovať jednotlivé funkcionality sú uvedené v metodike pre testovanie aplikácie.

1.3.2 Procesy manažmentu softvéru

Manažment verzií, manažment konfigurácií softvéru:

- **manažment verzií** – Po dokončení ucelenej funkcionality produktu je vytvorená nová verzia produktu, čo umožňuje prehliadanie zmien, ktoré boli vykonané, prípadne návrat k predchádzajúcej verzii ak je to nutné. Pre správu verzií používame nástroj GitHub. Bližšie informácie k používaniu verziovacieho nástroja GitHub sú uvedené v metodike verziovania.

1.3.3 Procesy vývoja a prevádzky softvéru

Manažment iterácií projektu

- **tvorba iterácie** – Iterácia projektu prebieha na každom tímovom stretnutí raz za týždeň. Na tímovom stretnutí prebieha plánovanie úloh, ktoré je potrebné spraviť. Plánovanie úloh pozostáva aj z identifikácie požiadaviek produktového vlastníka a upravovania požadovaných funkcií, ktoré ma obsahovať finálny produkt. Toto plánovanie prebieha formou diskusie,ktorej sa zúčastňujú všetci členovia tímu.

Manažment zberu požiadaviek:

- **riadenie požiadaviek na zmenu**: – Zákazníkom v našom projekte je produktový vlastník, ktorý svoje požiadavky na minimálnu funkcionalitu produktu definoval na začiatku vývoja. Nasledujúce požiadavky sú upravované a dopĺňané na tímových stretnutiach na základe tímových diskusií. V prípade, že nastane situácia kedy je potrebné zmeniť niečo, čo už bolo implementované, či už z dôvodu nejakej objavenej chyby alebo len nájdenie lepšieho riešenia, prípadne nutnosť upraviť existujúcu funkcionalitu kvôli závislosti novej funkcionality, platia rovnaké pravidlá ako pre vytváranie nových úloh. Na tímovom stretnutí prebehne diskusia, v ktorej sa preberú možné alternatívy a ak je to nutné, schváli sa nová úloha, ktorá bude riešiť požiadavku na zmenu.

Manažment testovania, manažment prehliadok:

- **vyhodnocovanie testov** – Vyhodnocovanie testov prebieha na dvoch úrovňach. Každý člen si píše svoje lokálne testy, pomocou ktorých si overí svoje riešenie. Zásadou je nezverejňovať kód, ktorý nefunguje. Pred integráciou novej funkcionality do aktuálnej verzie, je kód otestovaný aj na serveri a v prípade, že všetky testy neboli vyhodnotené úspešne, kód nie je integrovaný do aktuálnej verzie. Bližšie informácie o tom ako otestovať a integrovať kód do aktuálnej verzie sú uvedene v metodike kontinuálnej integrácie.
- **identifikácia a riadenie chýb v softvéri** – Vo vývoji softvéru používame testami riadený vývoj softvéru, kde najprv napíšeme testy a potom podľa nich píšeme kód, resp. implementujeme požadovanú

funkcionalitu. Každý člen je zodpovedný za testovanie svojho kódu. Ak však počas práce na svojej úlohe člen tímu objaví chybu, ktorá nebola dovtedy identifikovaná, poznačí túto chybu do nástroja pre manažment úloh ako úlohu do budúcna, označí ju ako chybu a pridá ju do zoznamu úloh. Na tímovom stretnutí je potom táto chyba prezentovaná ostatným a podľa závažnosti pridaná medzi úlohy, ktoré sa budú riešiť v ďalšom šprinte. Bližšie informácie o tom ako otestovať svoju funkcionalitu v backende alebo frontende sú uvedene v metodike testovania aplikácie.

– **technologická podpora jednotlivých činností** – Pre testovanie backend časti aplikácie používame nástroj Rspec pre Ruby on Rails, pre testovanie frontend časti aplikácie používame na testovanie funkcionality Jasmine

Manažment dokumentácie:

– **riadenie procesu dokumentovania** – Proces dokumentovania procesu vývoja pozostáva z niekoľkých častí, do ktorých patrí dokumentácia samotného riadenia procesu vývoja a inžinierske dielo. Inžinierske dielo obsahuje dokument technická dokumentácia, ktorá dokumentuje samotný kód. Za túto dokumentáciu je vo všeobecnosti zodpovedný člen tímu, ktorý bol zodpovedný za implementáciu vybranej funkcionality.

1.4 Sumarizácie šprintov

1.4.1 1. šprint

Prvý šprint sa zameriaval na organizačné záležitosti spojené s požiadavkami, ktoré na nás boli kladené v súvislosti s predmetom Tímový projekt a tvorbu základov pre ďalšie fungovanie projektu.

Ako issue tracker sme sa rozhodli použiť Jiru, kde sme si na začiatku šprintu vložili všetky úlohy na ktorých sme sa pred začatím šprintu dohodli.

Vytvorila sa webová stránka tímového projektu, repozitár pre samotný projekt na Githube, kde sa vytvorila základná štruktúra ako pre frontend klienta, tak aj backend server. Spolu s tým sa začala konfigurácia servera pre webovú stránku tímu a pre samotný projekt.

Základná funkcionalita projektu, na ktorej sa pracovalo zahŕňa prihlasovanie používateľov do systému, zobrazenie používateľovho dashboardu a následné manažovanie projektov(CRUD).

Nepodarilo sa nám úspešne ukončiť všetky úlohy v tomto šprinte. Za chybu považujeme to, že sme nesprávne odhadli náročnosť jednotlivých úloh. Všetky nedokončené úlohy sme sa rozhodli presunúť do ďalšieho šprintu. Ako je možné vidieť na *Burndown charte* mali sme naplánované úlohy za 19 story pointov no podarilo sa nám spáliť iba 13 story pointov.

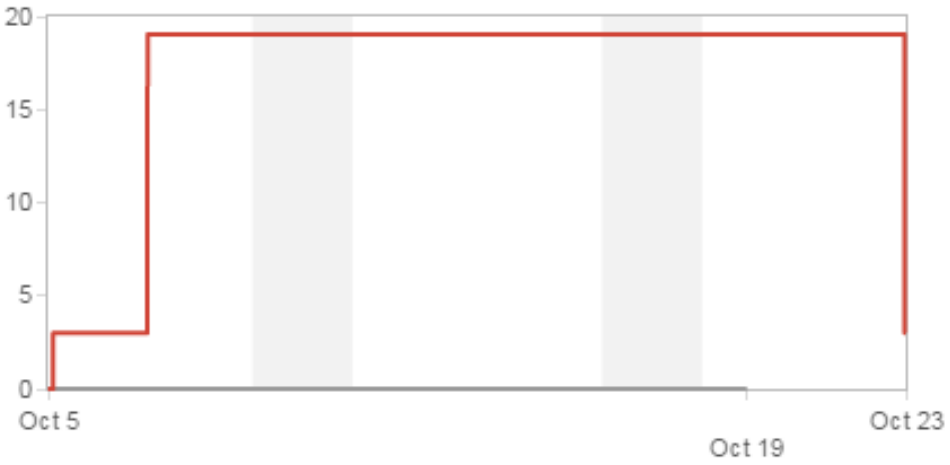
Identifikovali sme značné problémy s používaním verziovacieho systému, kde sme zaznamenali problém so zlým používaním vetiev a zlým vytváraním *commit message*-ov. Tieto aj ďalšie problémy sme zhrnuli v retrospektíve, ktorú môžeme vidieť nižšie.

Vyhnúť sa v budúcnosti

– Nevyvíjať nad „cudzou“ vetvou.

Pokračovať v tom ďalej

– Dodržiavať konvencie.



Obr. 1: Burndown chart.

– Udržiavať čitateľný kód.

Začať dodržiavať

– TDD - písať testy pred, nie po implementácií.
– Rozumne pomenovávať *commit message*.
– Napísať postup pre nové technológie, ktoré ešte neboli doteraz použité.
– Dohodnúť dopredu špecifikáciu pre komunikačné rozhranie, aby sa mohlo pracovať paralelne na viacerých úlohách.
– *Frontend mock-server*.

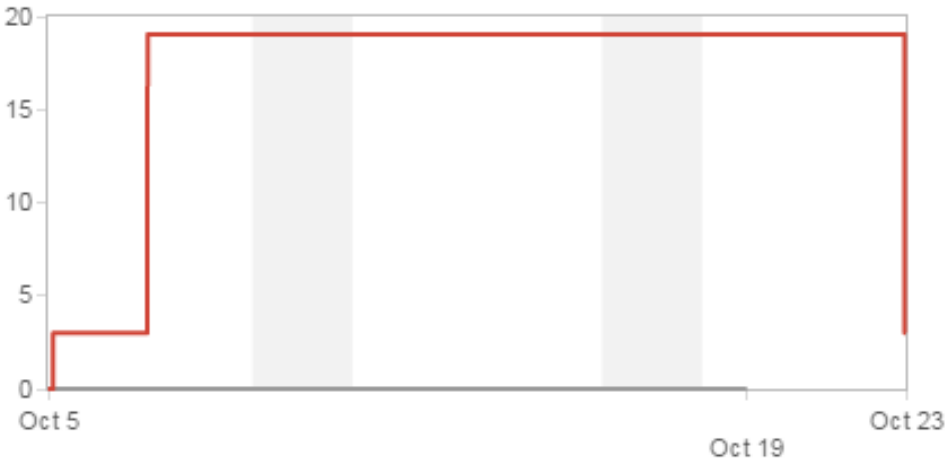
1.4.2 2. šprint

Najväčšia priorita v tomto šprinte sa kládla na ukončenie úloh z prvého šprintu. Spolu s tým, sme optimalizovali a vylepšovali už existujúci kód.

Rozhodli sme sa o tom, že sa chceme zúčastniť TP CUPu, kde sme si podali prihlášku a boli sme úspešne vybraný.

Vytvárali sa tu rôzne metodiky pre jednotlivé procesy, no kvôli nejasnostiam neboli všetky dokončené a boli tak presunuté do ďalšieho šprintu.

Kvôli pretrvávajúcim problémom s prihlasovaním používateľov pomocou prihlasovacích údajov s AIS(LDAP), sme sa rozhodli prestať venovať čas tomuto problému aby sme sa mohli venovať dôležitejším veciam.



Obr. 2: Burndown chart.

Ako je možné vidieť na *Burndown charte* mali sme naplánované úlohy za 20 story pointov no podarilo sa nám spáliť iba 12 story pointov.

Pri tomto šprinte sme identifikovali problém s Jirou, keď úlohy sa uzatvarali na konci šprintu a nie po ich vyriešení. Následne bola vytvorená nasledujúca retrospektíva.
Vyhnúť sa v budúcnosti

- Nenechávať si všetko na poslednú chvíľu.

Začať dodržiavať

- Začať používať Jiru počas práce na taskoch.

1.4.3 3. šprint

Dĺžka trvania tohto šprintu bola 3 týždne nakoľko sme sa v prvom týždni museli venovať tvorbe dokumentácie projektu a tvorbe metodík.

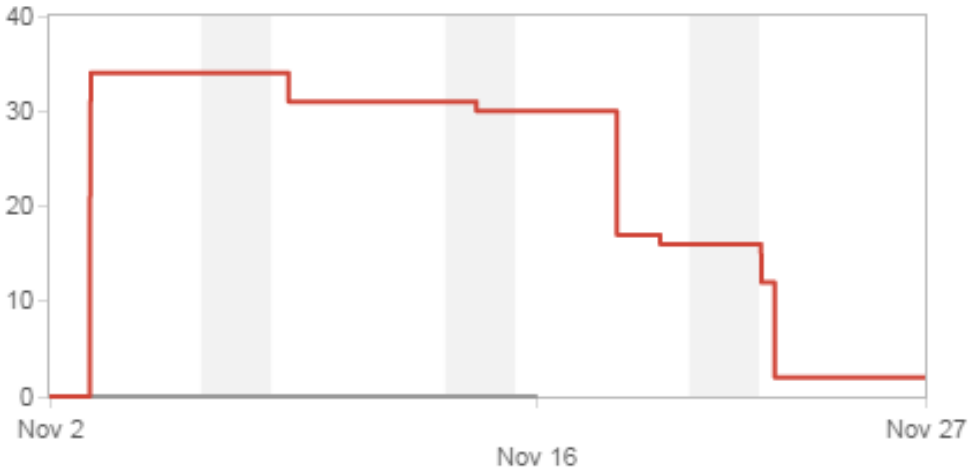
Zadefinovali sme si tu pojmy Mockup a Prototyp v našom projekte, nakoľko sme mali neustále problémy rozlišovať medzi nimi. Mockup predstavuje obrazovku s oblasťami záujmu, kde Prototyp je zoznam Mockupov.

Implementovala sa funkcionalita pre nahrávanie obrázkov, ktoré budú slúžiť na tvorbu Mockup-ov. Okrem toho sa ešte pracovalo na návrhu budúce dizajnu.
Ako je možné vidieť na *Burndown charte* mali sme naplánované úlohy za 34 story pointov no podarilo sa nám spáliť iba 29 story pointov. Tiež vidno, že úlohy už boli zatvarané ihneď po ich dokončení a nie až na konci šprintu, ako to bolo doteraz zvykom. Na konci sme si dali za cieľ priebežne pracovať na dokumentácií, čo sme aj zahrnuli do nasledujúcej retrospektívy.

Pokračovať v tom ďalej:

- Každý pracuje s vlastnou vetvou.

Začať dodržiavať:



Obr. 3: Burndown chart.

– Viac komunikovať s členmi tímu, ktorý už majú skúsenosti s daným problémom.

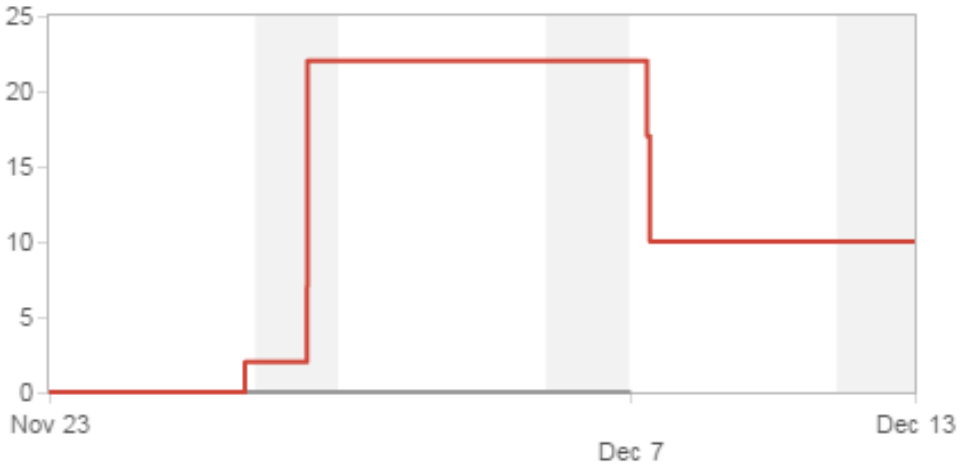
– Začať priebežne dopĺňať dokumentáciu.

1.4.4 4. šprint

Cieľom šprintu bolo umožniť vytváranie Prototypov a Mockupov. Ďalším cieľom bolo a vytvoriť štýlovanie z návrhov vytvorených v minulom šprinte, ktorý sa bude dať ďalej dobre škálovať.

Bol prácu s Mockupmi bol vytvorený editor na jeho editáciu. Aktuálne obsahuje iba základnú funkcionalitu na pridávanie nových objektov. Kvôli výskytu menších chýb sme sa rozhodli, že táto úloha bude presunutá do ďalšieho šprintu.

Návrh štýlovania je dokončené pre detail prehľadu projektu a je pripravené na ďalšie použitie.



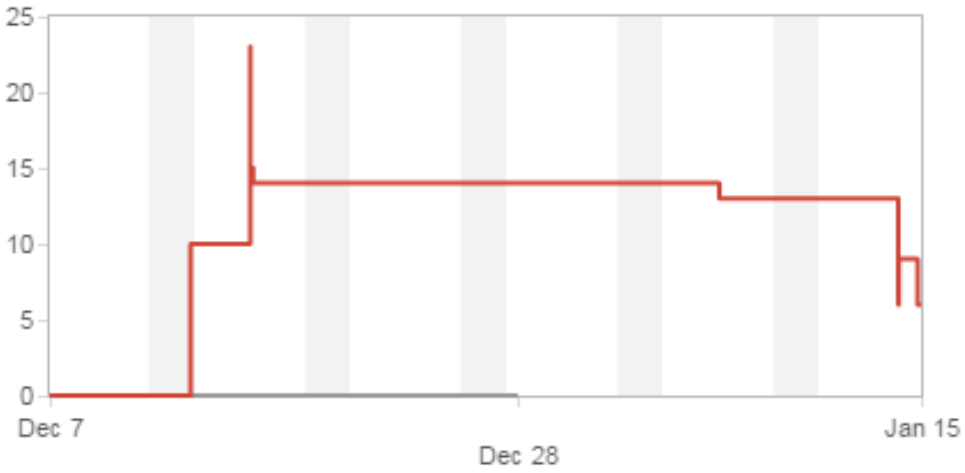
Obr. 4: Burndown chart.

Ako je možné vidieť na *Burndown charte* mali sme naplánované úlohy za 22 story pointov no podarilo sa nám spáliť iba 12 story pointov. V tomto šprinte nebola vytvorená retrospektíva.

1.4.5 5. šprint

Hlavnou úlohou tohoto šprintu bolo vytváranie a manipulácia Testov nad jednotlivými prototypmi. Ďalšou úlohou bolo vytvoriť návrh a následne podľa neho naštýlovať prehľad projektov.

Z minulého šprintu tiež ostalo menšie dopracovanie breadcrumbov v detailoch projektu a rovnako dôležité bolo aj opravenie menších chýb v Editore.

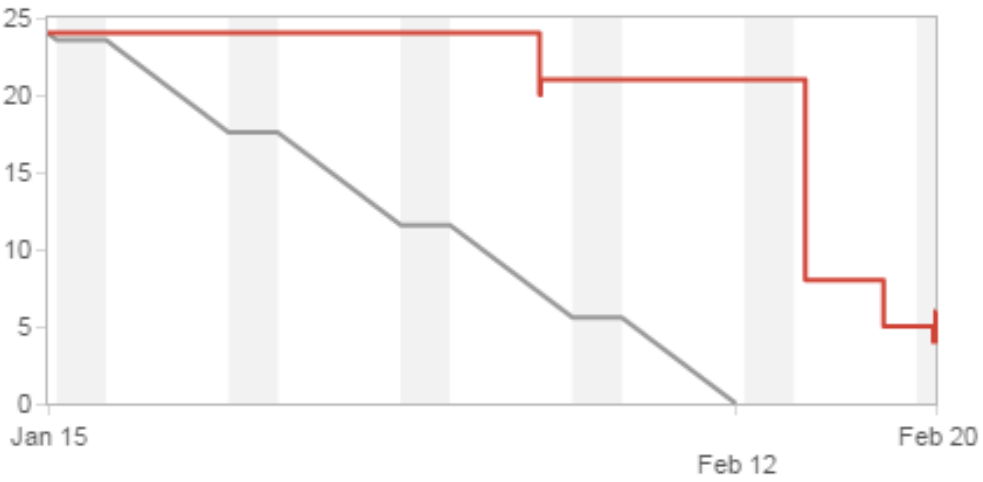


Obr. 5: Burndown chart.

1.4.6 6. šprint

Cieľom 6. šprintu bolo viacero dôležitých úloh. Jednou z nich bol rozsiahly refaktoring z dôvodu prístupových práv. Tiež sa pripravila validácia Testov podľa jednotlivých Taskov, ktoré obsahujú a vytvorilo sa zobrazenie vytvoreného testu pre používateľa.

V súvislosti so zobrazovaním Testov a ich následným spustením na testovanie používateľmi sa tiež riešilo vytvorenie webového linku pomocou, ktorého by vedel k testu pristupovať externý používateľ - tester.



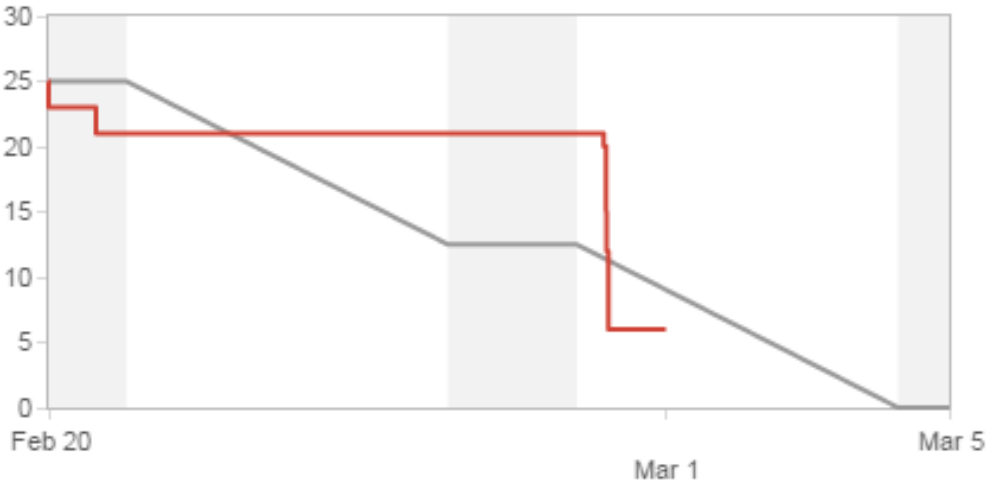
Obr. 6: Burndown chart.

1.4.7 7. šprint

V tomto šprinte sa nadväzovalo na spúšťanie testov. Hlavnou úlohou bolo zaznamenávanie akcií, ktoré tester pri testovaní vykonával - primárne zatiaľ iba jeho práca s myšou. S tým spojená úloha bola tieto zaznamenané akcie posielať na server, ktorý ich uloží do databázy.

Tiež sa nadväzovalo na Editor, v ktorom sa doplnili funkcie na pomenovanie elementov z dôvodu pridelenia oblasti záujmu pre Task. Jednotlivé elementy a Mockupy bolo potrebné poprepájať pomocou linkov, čo bola ďalšia úloha.Táto úloha bola rozpracovaná a bude sa na nej robiť aj naďalej na koľko s tým boli spojené viaceré problémy.

Bol vyriešený problém s prihlasovacou obrazovkou a tiež sa dokončilo celkové naštýlovanie tejto obrazovky.



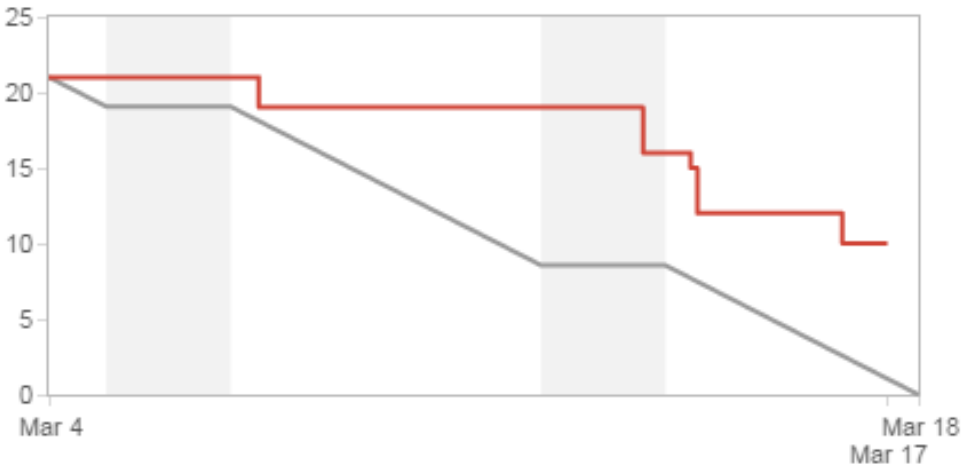
Obr. 7: Burndown chart.

1.4.8 8. šprint

Hlavným cieľom ôsmeho šprintu bolo dokončenie prehliadania Testu a jeho spustenia. S tým spojené bolo spustenie a uzavretie editácie Testu, jeho Taskov a Prototypov a sprístupnenie zdieľania linku na samotný Test.

V súvislosti s editorom bola dokončená funkcia prepojenia jednotlivých Mockupov Prototypu pomocou linkovacieho nástroja, ktorý elementom pridáva akcia na kliknutie. Okrem elementu, ktorý má priradený obrázok a reprezentuje tak časť Mockupu bol vytvorený prázdny element, ktorým je možné simulovať osobitný element.

Pre účely posielania rôznych udalostí v rámci našej aplikácie bol tiež vytvorený osobitný nástroj na prepojenie Angularu a čistého Javascriptu.



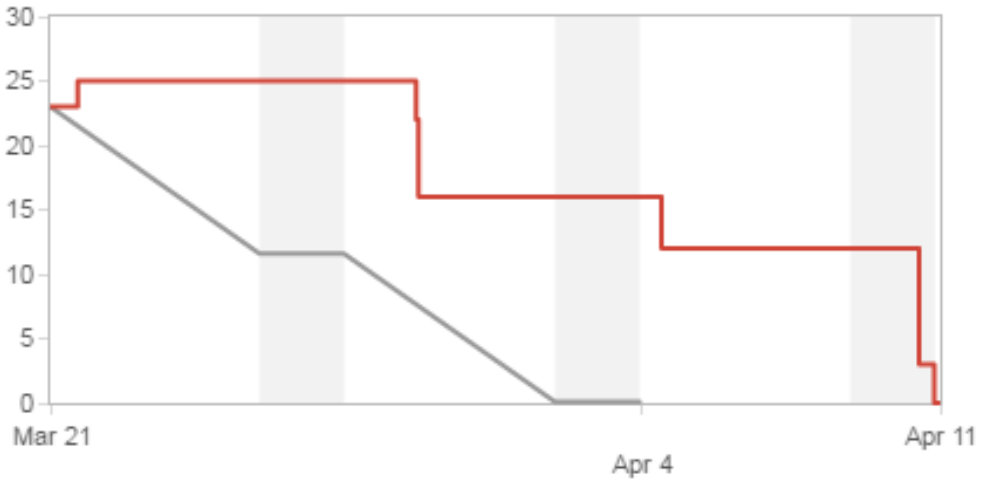
Obr. 8: Burndown chart.

1.4.9 9. šprint

Cieľom deviateho šprintu bolo zobrazovanie výsledkov Testu. Bolo dorobené vyhodnotenie akcií vykonaných testerom akými sú napríklad kliky myšou. Vyhodnotené výsledky boli Tvorcovi testu prezentované pomocou heat máp.

Po otestovaní Testu dostatočným počtom testerov je potrebné test uzatvoriť. Dorobilo sa ukončenie bežiacich testov vlastníkom Testu. Tiež sa dorobilo štýlovanie prehľadu Testov a prehľadu Projektov.

V rámci Editora sa dorobilo nastavenie šírky všetkých Mockupov Prototypu a ich vycentrovanie. Pridaná bola aj funkcionalita pre premenovanie a odstránenie Mockupov v Editore.

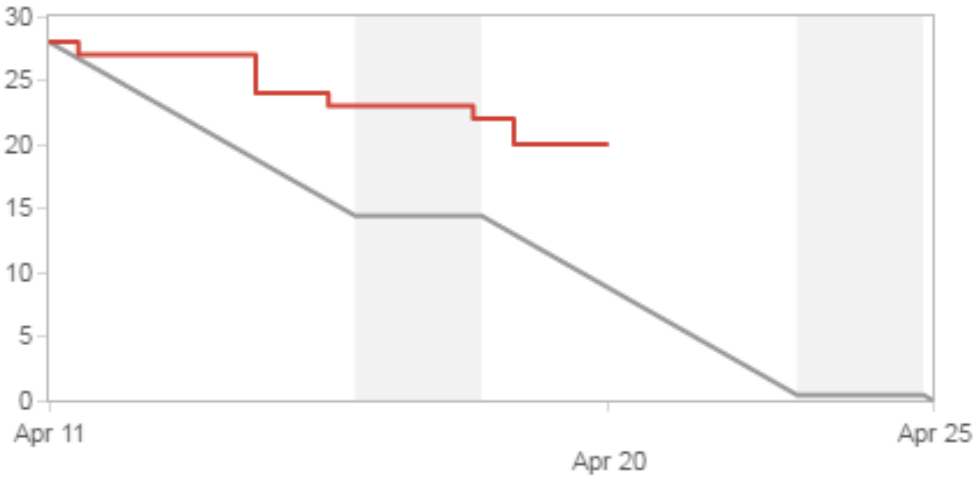


Obr. 9: Burndown chart.

1.4.10 10. šprint

Desiaty šprint bol zameraný na postupnú integráciu Eyetrackera na zaznamenávanie pohľadu testera. Táto úloha bola rozpracovaná a pracuje sa na nej ďalej. Rovnako bolo rozpracované zobrazovanie výsledkov testov.

Editor bol podrobený rozsiahlemu preštýlovaniu. Bol vytvorený návrh a tiež rozrobená samotná implementácia do aktuálneho editora, na ktorej sa bude ešte ďalej pracovať.

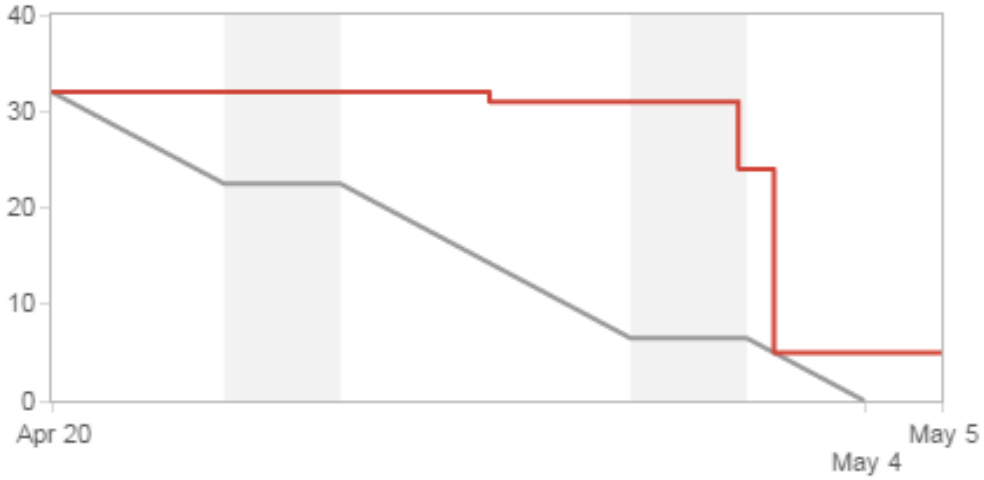


Obr. 10: Burndown chart.

1.4.11 11. šprint

Cieľom tohoto šprintu bolo dokončenie jednotlivých úloh do prezentácie na TP cup. Najdôležitejšou úlohou bolo dokončenie integrácie Eyetrackera a zobrazovanie výsledkov sledovania pohľadu testera pri vykonávaní testu. Tiež sa okrem klikov myši dorobil aj jej pohyb, ktorý je možné spolu so sledovaním pohľadu sledovať na prehrávaní priebehu testu.

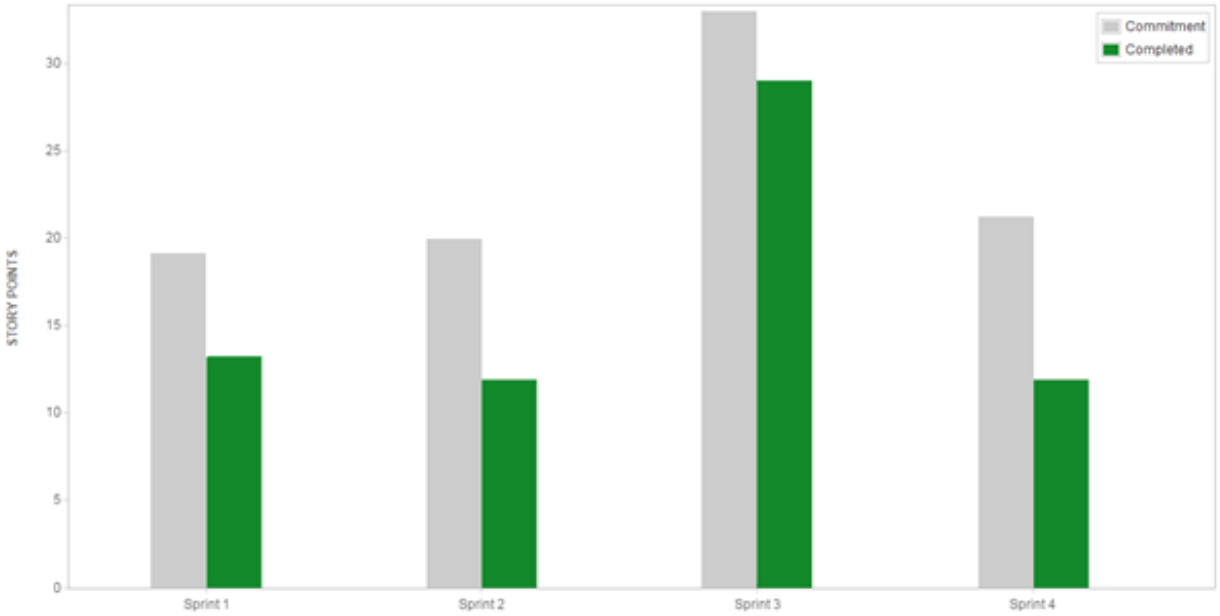
Dokončili sa prípravy na IITSRC a TP cup, ktoré záhŕňali vytvorenie plagátu a prípravy prezentačného projektu. Tiež sa dokončilo štýlovanie editora a opravili sa niektoré chyby.



Obr. 11: Burndown chart.

1.4.12 Vyhodonotenie šprintov

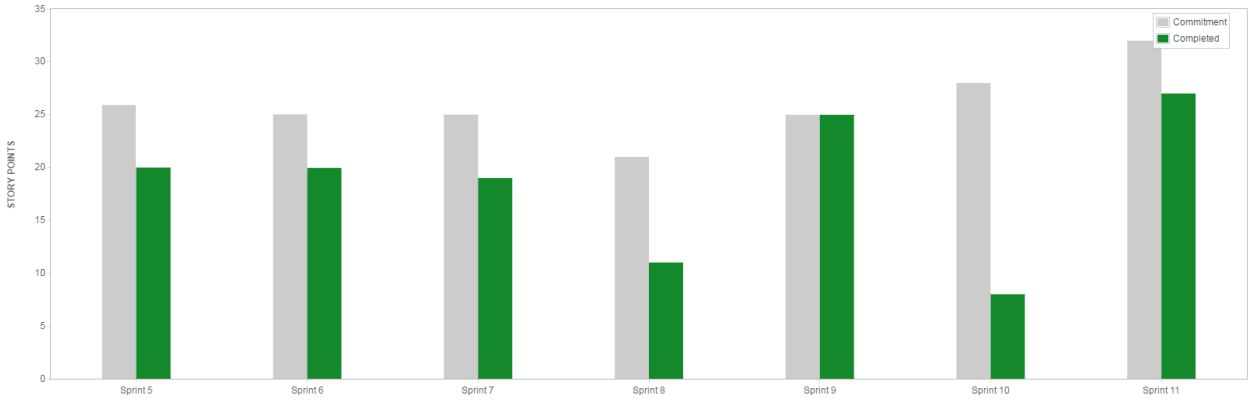
Po ukončení každého šprintu sme si vyhodnotili úspešnosť daného šprintu, teda nakoľko sa nám podarilo splniť si úlohy, ktoré sme si zadali na začiatku šprintu. Nasledujúci graf ukazuje pre jednotlivé šprinty, koľko story pointov sme mali naplánovaných a koľko sa nám ich reálne podarilo spáliť.



Obr. 12: Velocity chart - 1. semester

Môžeme vidieť, že sa nám darí spaľovať 12 story pointov za šprint v priemere. Toto spaľovanie nemá výraznú stúpajúcu alebo klesajúcu tendenciu z čoho môžeme usúdiť, že sa nám aktuálne nedarí zlepšovať sa.

Tretí šprint tvorí značnú odchýlku, ktorá je ale zapríčinená tvorbou dokumentácie a metodík, ktoré mali pridelenú značnú časť story pointov. Ak si odmyslíme tieto úlohy, dostaneme sa na úroveň spálenia 11-tich story pointov.



Obr. 13: Velocity chart - 2. semester

Ako môžeme vidieť, v druhom semestri došlo k značnému zlepšeniu v spaľovaní story pointov, kedy sme boli schopný spáliť približne 20 story pointov.

Výnimku tvoria šprinty 8 a 10, kde sme spálili iba 11 a 8 story pointov. To bolo ale zapríčinené veľkou náročnosťou úloh v daných šprintoch, ktoré sme museli prenášať do ďalšieho šprintu, kde sme si ale pridali o to menej nových úloh.

1.5 Používané metodiky

V rámci vývoja a tvorby nášho softvérového produktu sme vypracovali niekoľko metodík, ktoré opisujú konkrétne procesy v našom prostredí, ktoré počas práce na projekte dodržiavame. V tejto kapitole sa nachádza prehľad použitých metodík, ako aj stručný prehľad o čom pojednáva daná metodika.

1.5.1 Metodika verziovania

Na správu verzií používame verziovací systém Git. V tejto metodike sa nachádzajú základné pravidlá a konvencie aké správy dávať do commitov, kedy commitovať, ako pomenovať jednotlivé branche, kedy vytvárať nové branche a ako postupovať pri mergovaníbranchí.

1.5.2 Metodika revízie kódu

Táto metodika hovorí o tom, aké knižnice používať na kontrolu správnosti nášho kódu, a taktiež ako môžeme tieto nástroje nainštalovať do vývojového prostredia. Revízia kódu je veľmi dôležitá, aby sme dokázali udržiavať náš kód jednotný a upravený, ale aj aby sme predchádzali možným chybám, ktoré mohli vzniknúť pri vývoji.

1.5.3 Metodika kontinuálnej integrácie

V tejto časti sa nachádza presný postup ako používať nástroj Wercker na účely kontinuálnej integrácie, ale taktiež aj zoznam závislostí, respektíve knižníc, ktoré sú potrebné na spustenie. Pomocou kontinuálnej integrácie udržiavame funkčný kód neustále nasadený a hlavne tento kód je otestovaný backendovými aj frontendovými testami.

1.5.4 Metodika pre správu riadenia projektu a požiadaviek

Pojednáva o používaní softvérového nástroja JIRA na účely spravovania riadenia projektu a požiadaviek. Obsahuje základné pravidlá ako pracovať s nástrojom JIRA.

1.5.5 Metodika pre písanie kódu v JavaScripte

Obsahuje konvencie, ktoré sa dodržiavajú pri písaní kódu na frontendovej časti aplikácie, čiže písanie JavaScriptu, ale hlavne pravidlá pre písanie vo frameworkuAngularJS. Nachádzajú sa tu pravidlá ako pomenovávať jednotlivé triedy, aká je súborová štruktúra aplikácie, ako komentovať a dokumentovať kód, ale v neposlednom radeaký štýl kódu používať pri programovaní.

1.5.6 Metodika pre písanie kódu v Ruby on Rails

Táto časť podobne ako metodika pre písanie kódu v JavaScripte obsahuje konvencie ako písať kód na backendovej strane, aké sú konvencie pre pomenovanie premenných, respektíve tried, ako komentovať a dokumentovať kód.

1.5.7 Metodiky pre testovanie aplikácie

V týchto metodikách sa nachádzajú postupy ako testovať jednotlivé časti aplikácie, či už sa jedná o backendovú alebo frontendovú časť. Použité knižnice na testovanie sú RSpec (backend) a Jasmine (frontend). Sú tu pravidlá ako písať jednotlivé testy, a pre ktoré časti kódu písať testy.

1.6 Globálna retrospektíva

V priebehu semestra sme si zadefinovali niekoľko procesov z rozličných oblastí, ktoré nám umožňovali pracovať koordinovane a efektívne. Keďže nami využívaný vývoj je založený na metodológii SCRUM, kde základným prvkom sú iterácie, aj k oblasti riadenia sme pristupovali iteratívnym spôsobom. Počas celého semestra sme kládli dôraz na to, aby sa fungovalo presne podľa stanovených procesov a taktiež sme sa snažili tieto využívané procesy neustále zlepšovať.

Po ukončení jednotlivých šprintov, sme si vždy daný šprint vyhodnotili a vytvorili k nemu retrospektívu. V rámci tejto retrospektívy mal každý člen tímu možnosť vyjadriť sa k priebehu daného šprintu. Primárnym cieľom retrospektívy bolo poukázať na procesy, ktoré sa nám overili ako dobré a chceme ďalej pokračovať v ich využívaní, ale taktiež aj poukázať na procesy ktoré nefungujú správne a chceme sa im v budúcnosti vyhnúť, prípadne ich vylepšiť.

Medzi základné procesy, ktoré sa nám podarilo dobre nastaviť a umožňujú nám pracovať efektívne a koordinovane patria napríklad:

- vývoj riadený testami (TDD) - písať testy pred a nie po implementácií. Danú funkcionalitu pokladáme za hotovú až v prípade, že sú adekvátne otestované všetky jej potrebné časti. Tento proces patrí medzi najzákladnejšie procesy a preto kladieme vysoký dôraz na jeho dodržiavanie. Keďže ide pre nás o novú paradigmu, kladieme aktuálne väčší dôraz na to, či sú testy napísané a nie kedy boli napísané. Do budúcna je však plánom neustále sa zlepšovať a pracovať na tom, aby naozaj išlo o vývoj riadený testami a teda testy boli napísané pred implementáciou.
- verziovanie kódu - zadefinovali sme si metodiku ako efektívne pracovať s verziami kódu, aby bol vývoj jednotlivých komponentov alebo funkcionalít prehľadný a ľahko organizovateľný. Taktiež kladieme vysoký dôraz na to, kedy je nový komponent alebo funkcionalita považovaná za hotovú. Pred pridaním novej funkcionality do hlavnej vetvy musí autor vytvoriť pull request a následne prejde táto funkcionalita kontrolou kompetentných členov tímu a až po akceptovaní všetkými kompetentnými členmi je táto funkcionalita pridaná do hlavnej vetvy.
- kvalita kódu - počas vývoja kladieme vysoký dôraz na kvalitu kódu. Dbáme na to, aby bol kód prehľadný, dodržiaval stanovené konvencie a taktiež aby boli potrebné časti dostatočne zdokumentované.

Taktiež sme identifikovali niekoľko nových procesov, ktoré by sme postupne chceli začať dodržiavať:

- definovať metodiku pre nové technológie a procesy, ktoré ešte neboli využívané - je vhodné mať pre všetky využívané technológie a procesy dobre zadefinované metodiky, aby mohli všetci členovia tímu efektívne využívať tieto technológie a dodržiavať stanovené procesy.
- pravidelnejšie aktualizovať svoju činnosť v nástroji pre manažment úloh - aktualizácia aktivity v nástroji manažmentu úloh aktuálne prebieha viacmenej nárazovo pri ukončovaní šprintu. Je vhodné svoju aktivitu aktualizovať pravidelne, aby mali všetci členovia tímu prehľad, kto na čom aktuálne pracuje.

Počas vývoja sme narazili aj na problémy, ktorým by sme sa v budúcnosti chceli vyhnúť:

- nevyvíjať v rámci vývojovej vetvy niekoho iného - aby bol vývoj dobre organizovaný, je potrebné klásť veľký dôraz na dodržiavanie metodík verziovania kódu
- nenechávať si všetko na poslednú chvíľu - je potrebné pracovať viac pravidelnejšie a menej nárazovo

Na začiatku semestra sme si zadefinovali MVP pre zimný semester (kapitola 2.2.1 - dokument Inžinierske dielo). Počas celého semestra sme usilovne pracovali na tom, aby sa nám podarilo stanovené MVP implementovať. Vzhľadpm k časovému obmedzeniu semestra a pomerne dlhej inicializácií, kde sme sa snažili dobre zadefinovať všetky potrebné procesy a metodiky, aby sme následne mohli fungovať efektívne a koordinovane, sa nám nepodarilo stanovené MVP kompletne implementovať.

Konkrétne sa nám nepodarilo implementovať nasledovnú funkcionalitu:

- zadefinovanie oblastí záujmu (ide o pomerne komplexnú a náročnú funkcionalitu, ktorá je aktuálne v štádiu vývoja)

- získanie odkazu na test (uskutočnenie testu)
- vykonanie testu testerom
- sledovanie aktivity testera
- automatická analýza zobieraných dát

V aktuálnej podobe náš systém ponúka možnosť registrácie používateľov, pričom zatiaľ systém nie je delený na používateľov podľa ich rôl. Aktuálne sa v systéme nachádza iba jedna rola používateľov a to zadávateľ testu. Po registrácií má používateľ možnosť vytvárať si projekty, v rámci ktorých môže vytvárať testy a špecifikovať ich jednotlivé úlohy. Pri definovaní úlohy testu si používateľ zvolí, nad ktorým prototypom (ktorý aktuálne predstavuje iba obrázok, ktorý môže používateľ nahrať do systému) sa bude úloha vykonávať. Taktiež má používateľ možnosť vytvárať si prototypy, nad ktorými sa budú testy vykonávať pomocou interaktívneho editora, ktorý bude taktiež slúžiť na definovanie oblastí záujmu. Avšak tento editor je aktuálne vo verzii prvého prototypu, ktorý poskytuje iba základnú funkcionalitu a to otvorenie obrázku a manipuláciu s ním (jeho premiestňovanie a zmena veľkosti).

2 Inžinierske dielo

2.1 Úvod

Tento dokument slúži ako technická dokumentácia k predmetu Tímový projekt. Opisuje sa v ňom práca na projekte na tému Testovanie používateľského zážitku na webe, v rámci ktorej sme vytvorili platformu pre online UX testovanie spolu so sadou základných UX nástrojov na testovanie webových stránok a prototypov.

V kapitole 2.2 opisujeme ciele projektu. Tieto ciele sa zameriavajú na minimálnu funkcionalitu, ktorú sme si zaumienili implementovať ešte počas zimného semestra tak, aby sme mali funkčný prototyp, ale taktiež sú v tejto kapitole zahrnuté aj ciele pre letný semester.

Kapitola 2.3 sa sústreďuje na architektúru nášho systému, kde poskytujeme globálny pohľad na náš systém. Jednotlivé moduly, komponenty a vzťahy medzi nimi sú tu detailne popísané a zobrazené na diagramoch ako na vyššej tak aj na nižšej úrovni pre frontend a backend.

Nasledujúca kapitola ďalej rozvíja kapitolu architektúry, kde podrobnejšie popisuje najdôležitejšie moduly, komponenty a iné dôležité časti systému.

2.2 Ciele projektu

Hlavné ciele nášho projektu sú:

- navrhnúť a implementovať platformu pre online UX testovanie
- navrhnúť a implementovať sadu základných UX testovacích nástrojov
- realizovať prepojenie našej platformy so sledovaním pohľadu
- umožniť zapojenie davu do online testovania

Náplňou nášho projektu je vytvorenie platformy pre online UX testovanie, ktorá bude spájať tradičné a online UX testovanie. Táto platforma bude pozostávať zo sady základných UX nástrojov, pomocou ktorých bude možné testovať webové stránky v rozličných podobách. Pričom aj zo statických prvkov ako napríklad screenshot bude možné vytvoriť klikateľný prototyp. Platforma bude poskytovať možnosť testovania webových stránok vo forme:

- screenshotov – grafický návrh webu, alebo screenshot webu
- mockupov – low fidelity návrh
- live webov – plne funkčný web

Tvorca testov bude mať možnosť vytvárať rozličné scenáre testov a zvoliť si špecifický segment používateľov, ktorý bude predstavovať testovaciu vzorku. Špecifickým segmentom používateľov, je členenie podľa rozličných demografických údajov (pohlavie, vek a pod.), alebo podľa záujmov používateľov. Tvorca testu si taktiež bude môcť zvoliť požadovnú veľkosť testovacej vzorky a žiadané výstupy z testu, t.j. aká aktivita bude sledovaná u účastníkov testu, a aké analýzy budú vykonané nad zozbieranými dátami. Bude možné sledovať základnú aktivitu používateľov, ako napríklad:

- splnenie/nesplnenie úlohy
- čas trvania jednotlivých akcií

- pohyb myši
- kliknutia
- vstup z klávesnice

Následne bude možné nad zozbieranými dátami vykonávať rozličné analýzy, ako napríklad:

- štatistika úspešnosti
- štatistika doby trvania jednotlivých úloh
- čas do prvého kliknutia
- počet kliknutí
- sledovanie AOI (oblastí záujmu):
  - čas do prvej návštevy
  - počet návštev
  - počet klikov
- generovanie teplotných máp klikov
- generovanie teplotných máp scrollovania

Platforma bude taktiež umožňovať sledovanie pohľadu účastníkov testu, vďaka čomu bude možné detailnejšie zaznamenávať aktivitu používateľov a následne vykonávať analýzy aj nad týmito dátami. Realizácia sledovania pohľadu bude prebiehať pomocou prepojenia na UX platformu dostupnú u nás na fakulte. Práve sledovanie pohľadu obohatí našu platformu o funkcionalitu, ktorá v doterajších nástrojoch pre online UX testovanie nie je dostupná.

Účastníci budú k testu prideľovaní priamo našou platformou, alebo bude možné prizvať účastníkov k testu pomocou jednoduchého URL odkazu. Distribúciu testov medzi používateľov bude potenciálne možné riešiť aj pomocou systému Crowdex, vyvíjaného u nás na fakulte. Platforma bude umožňovať paralelné vykonávanie špecifického testu a vďaka tomu bude možné do daného testu zapojiť veľké množstvo účastníkov a zobierať tak veľké množstvo dát.

Platforma bude implementovaná tak, aby bola modulárna a ľahko rozšíriteľná. Vďaka modulárnosti platformy ju bude možné neustále rozširovať o novú funkcionalitu a oddelením backendu a frontendu (teda realizáciou backendu iba ako API platformy) sa umožní využitie tejto API aj v iných projektoch.

2.2.1 Zadefinovaný MVP

V rámci našej práce na projekte počas zimného a letného semestra sme si ako cieľ stanovili vytvoriť počas zimného semestra funkčný prototyp, ktorý bude nasadený online a bude poskytovať nasledovnú funkcionalitu:

- Možnosť registrácie používateľov
- Zadávateľ testu musí mať možnosť:
  - vytvoriť nový projekt
  - vytvoriť test
  - nahrať obrázok

    - zadefinovať úlohy (vytvorenie scénaru)
    - zadefinovanie oblastí záujmu
    - získanie odkazu na test (uskutočnenie testu)
- Tester:
    - po kliknutí na odkaz od zadávateľa testu sa prihlási a vykoná predefinované úlohy
- Aplikácia musí sledovať nasledovnú aktivitu testera
    - splnenie / nesplnenie úlohy
    - pohyb myši
    - kliknutia
    - čas trvania úloh
- Aplikácia poskytne zadávateľovi testu po jeho ukončení výslednú analýzu v podobe metrík
    - štatistika splnenia / nesplnenia úlohy
    - časy trvania úloh
    - metriky nad oblasťami záujmu
        - čas do prvého kliknutia
        - čas do prvej návštevy
        - počet kliknutí

2.2.2 Vytvorený prototyp

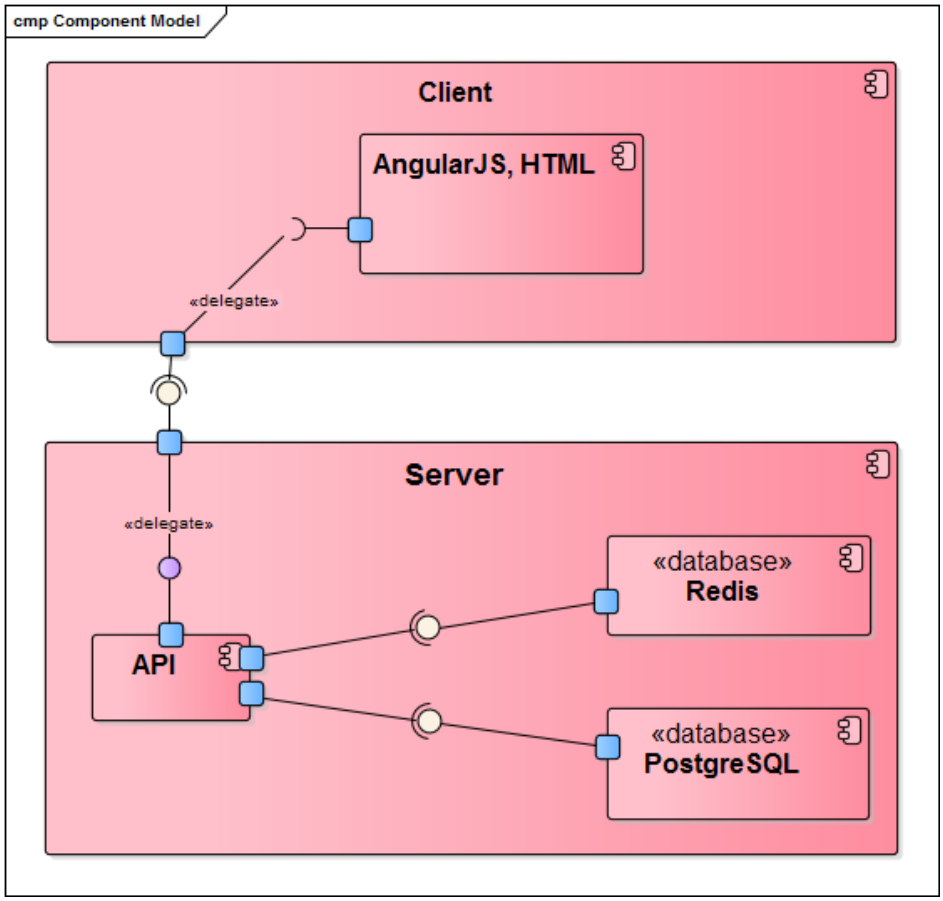
V rámci práce na projekte počas letného a zimného semestra sa nám podarilo vytvoriť funkčný prototyp, ktorý spĺňa väčšinu funkcionality, ktorá bola stanovená ako MVP a taktiež poskytuje dodatočnú funkcionalitu, ktorá bola vytvorená nad rámec zadefinovaného MVP. Funkčný prototyp v aktuálnom stave poskytuje nasledovnú funkcionalitu:

- Možnosť registrácie používateľov
- Funkcionalita poskytnutá zadávateľovi testu:
    - vytváranie projektov
    - vytváranie testov
    - vytváranie taskov (úloh) v rámci testov (definovanie scénaru testu)
    - nahrávanie obrázkov
    - vytváranie klikateľných prototypov pomocou interaktívneho editora
    - definovanie ukončovacej podmienky v rámci jednotlivých úloh (definovanie na ktorý element má byť posledný klik aby bola úloha považovaná za splnenú)
    - získanie zdieľateľného odkazu na vytvorený test
- Funkcionalita poskytnutá testerovi:
    - po kliknutí na odkaz testu možnosť vykonať test
- Aplikácia v rámci vykonávania testu sleduje a zaznamenáva:
    - splnenie / nesplnenie úohy

  - pohyb myši
  - kliknutia
  - čas trvania jednotlivých úloh
  - pohľad používateľa
  - údaje o testerovi (prehliadač, operačný systém, rozlíšenie)
- Aplikácia poskytuje používateľovi rozličné výsledky na základe vykonaných testov:
  - štatistiky splnenia jednotlivých úloh
  - štatistiky časov trvania úloh
  - teplotné mapy klikov (všetky kliky alebo prvé kliknutia)
  - teplotné mapy pohybu myši
  - teplotné mapy pohľadu používateľov
  - možnosť prehratia videa, na ktorom je znázornený pohyb myši špecifického používateľa
  - možnosť prehratia videa, na ktorom je znázornený pohľad špecifického používateľa
  - graf prechodov medzi podstránkami webu počas vykonávania úlohy

2.3 Architektúra

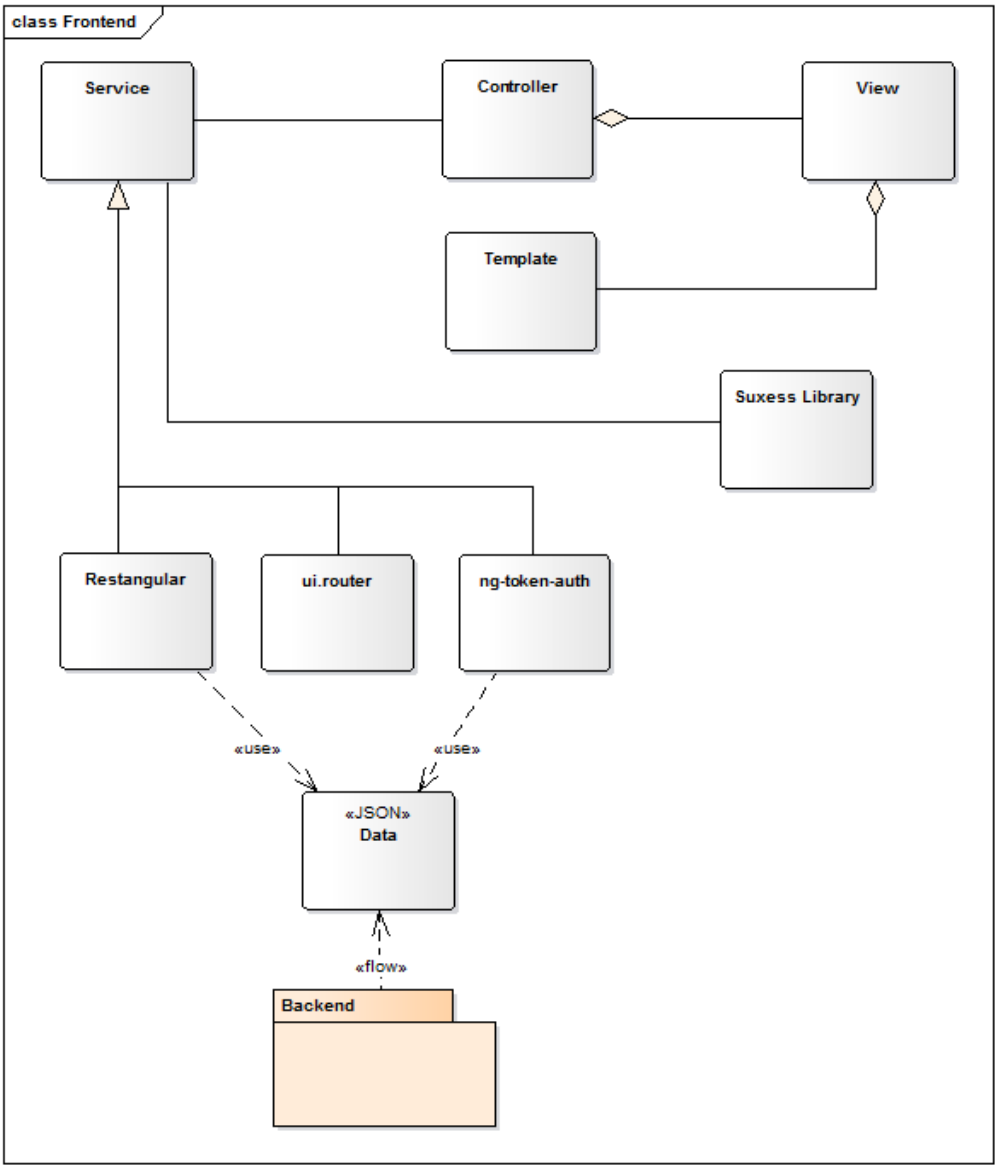
Architektonický štýl klient–server vyjadruje architektúru, v ktorej má aplikácia dve časti – klientskú časť a serverovú časť. Klientská časť je prezentovaná používateľovi a všetky dáta žiada od servera, kým serverová časť, ktorá je reprezentovaná ako API, vykonáva aplikáčnú logiku a pristupuje k databáze. V našom projekte používame tento štýl, pretože vytvárame webovú aplikáciu v HTML a Javascripte, ktorá má aplikáčnú logiku realizovanú na serveri v Ruby on Rails. Serverová časť reprezentovaná ako API využíva dva typy databázy. V databáze PostgreSQL sú uložené údaje reprezentované dátovým modelom. Databáza Redis sa využíva pri vyhodnocovaní metrík pre vykonané testy.



Obr. 14: Architektúra systému.

2.3.1 Frontend

Na obr. č. 15 je znázornený diagram komponentov frontendu znázorňujúci architektúru. Využívame knižnicu Angularjs a tomu je aj prispôsobená architektúra. Zobrazovaná stránka je reprezentovaná triedov View, ktorá v sebe agreguje html template a je agregovaná controllerom. Keďže sa snažíme o tenký controller, logiku aplikácie zabezpečujú servisy a backend api. Vzťah servisov a backendu je znázornený napríklad medzi restangularom, ktorý slúži na rest požiadavky a api – ruby on rails.

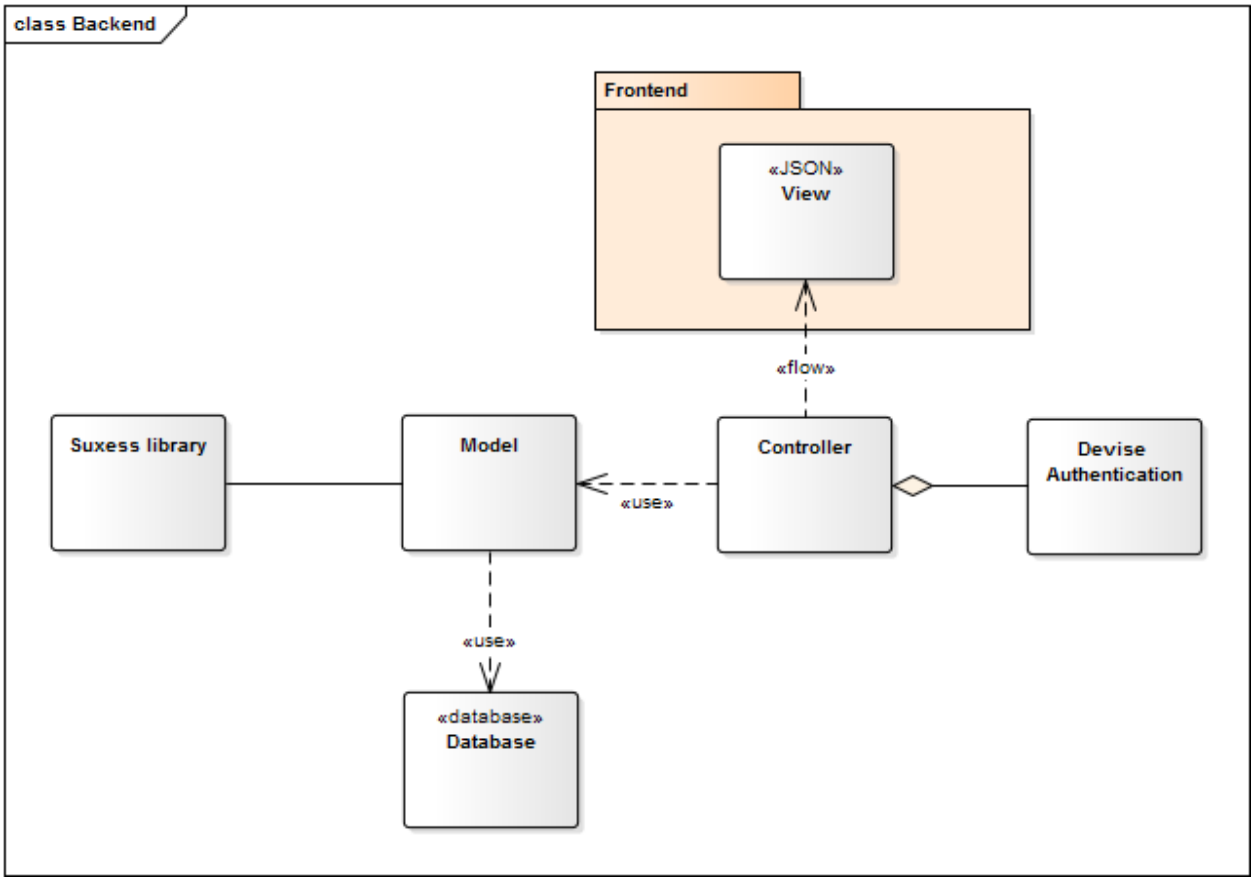


Obr. 15: Diagram komponentov frontedu.

2.3.2 Backend

Na obr. č. 16 je zobrazený diagram komponentov backendu, ktorý zobrazuje architektúru. Backend je realizovaný ako API pomocou webového frameworku Ruby on Rails. V implementácii sa dodržiavajú základné konvencie architektonického štýlu MVC (Model-View-Controller). Frontend je oddelene realizovaný pomocou frameworku Angular, takže časť View nie je na backende potrebná. Frontend komunikuje s backendom pomocou REST API, ktoré je implementované pomocou kontrolerov. Kontroler komunikuje s príslušným modelom, v ktorom sa nachádza logika pracujúca s dátami. Dáta sú uchovávané v databáze PostgreSQL.

Backend bude obsahovať našu vlastnú knižnicu *Suxess Library*, ktorá bude slúžiť na spracovanie dát získaných zo záznamov aktivít používateľov, realizovanie výpočtov nad týmito dátami a vyhodnocovanie testov.



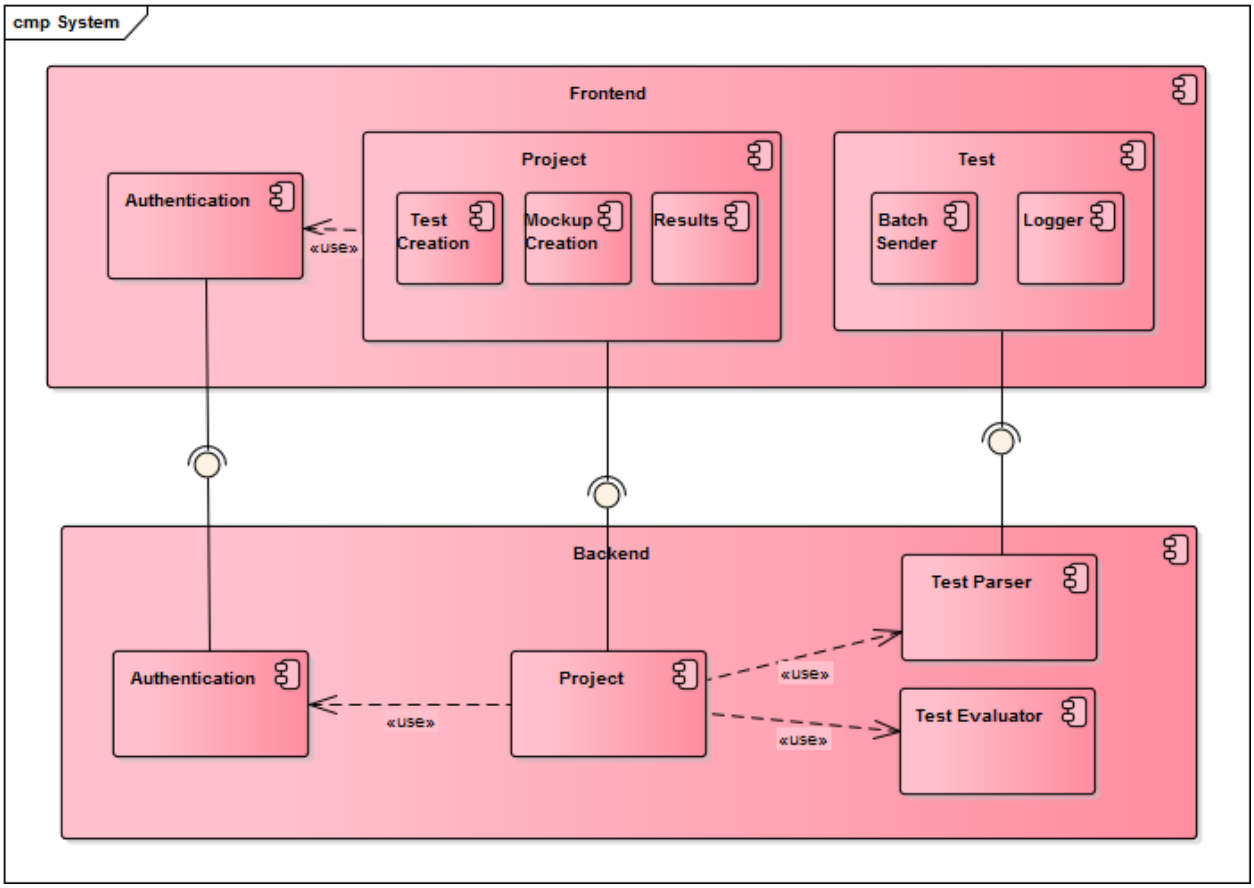
Obr. 16: Diagram komponentov backendu.

2.3.3 Komponenty systému

Na diagrame na obr. č. 17 je znázornený diagram komponentov systému. Opisuje komponenty, ktoré sme doteraz identifikovali a ich komunikáciu.

Na frontende existujú tri komponenty. *Autentifikácia*, ktorá komunikuje s rovnomenným komponentom na backende a zabezpečuje autentifikáciu používateľa. Tento komponent využíva aj *Project*, ktorý ho potrebuje, aby poznal používateľa a jeho práva. Komponent *Project* sa stará o vytváranie testov a mockupov ako aj o zobrazovanie výsledkov. Komponent *Test* slúži na testovanie mockupov používateľmi a využíva *Logger* na sledovanie používateľa a *Batch Sender* na posielanie záznamov z testovania.

Na backende sú komponenty, ktoré zabezpečujú API pre frontend. Komponent *Project* zabezpečuje CRUD, ktorý je potrebný na frontende. Komponent *Test Parser* slúži na spracovávanie a ukladanie výsledkov testovania v reálnom čase. *Test Evaluator* slúži na ich následne vyhodnocovanie.



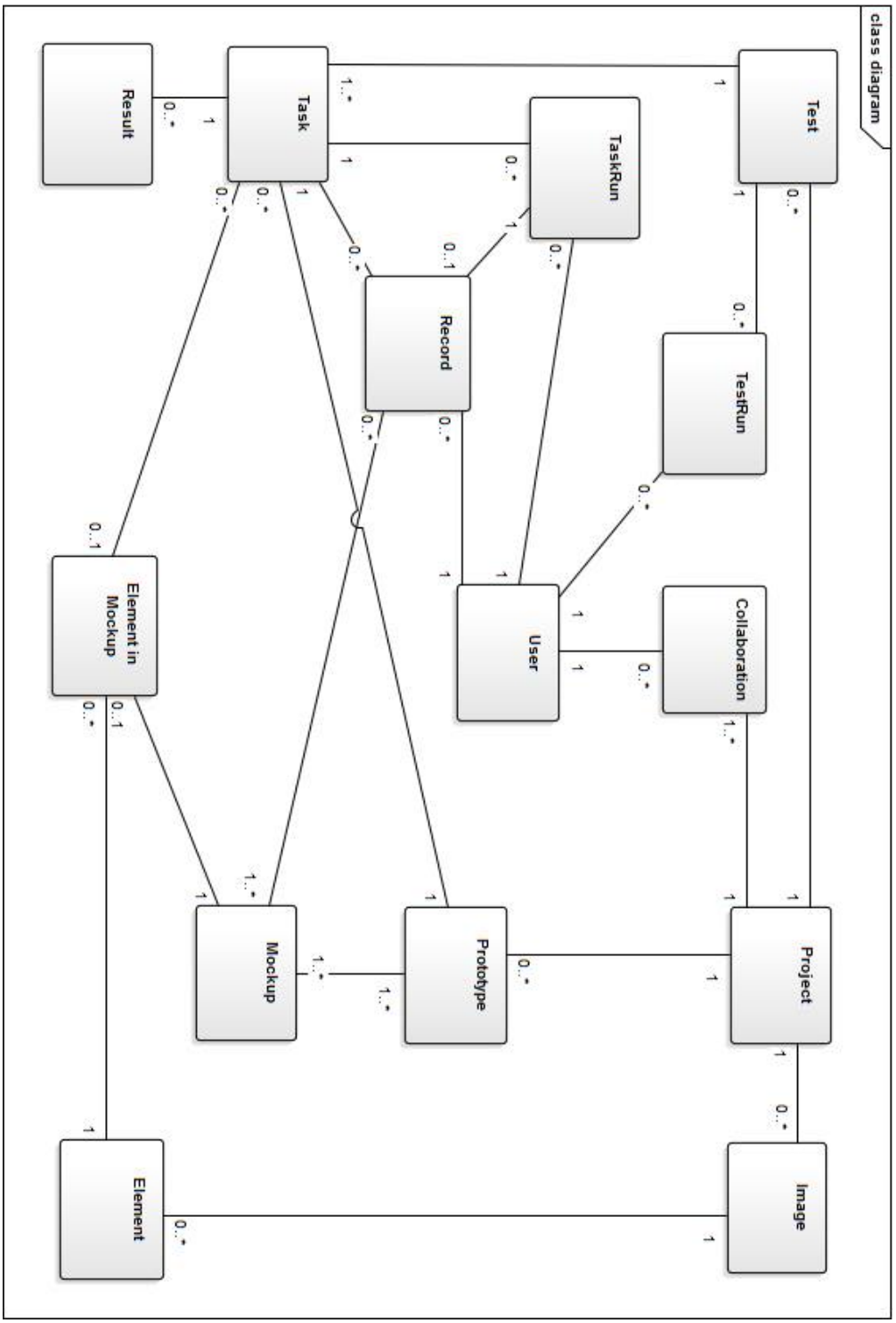
Obr. 17: Diagram komponentov systému.

2.3.4 Dátový model

Na obr. č. 18 je znázornený diagram dátového modelu nášho projektu. Entita *User* predstavuje používateľa našej aplikácie. Používateľ má možnosť vytvoriť si projekt (entita *Project*), pričom jeden používateľ môže vytvoriť viacero projektov a na jednom projekte môže spolupracovať viacero používateľov. Tento vzťah je realizovaný pomocou prepojovacej entity *Collaboration*, v ktorej je určená rola používateľa na konkrétnom projekte. Projekt predstavuje testovaný produkt. V rámci projektu môže používateľ vytvárať testy (entita *Test*), ktoré sa skladajú z menších úloh (entita *Task*). Entita *User* taktiež predstavuje testera, ktorý vykonáva testy. Každé uskutočnenie testu používateľov zaznamenávame v entite *TestRun*, pričom samotne každá vykonaná úloha používateľom je zaznamenaná v entite *TaskRun*.

Každá úloha je vykonávaná nad konkrétnym prototypom (entita *Prototype*), ktorý vytvára tvorca projektu. *Prototype* je zložený z podstránok prezentovaných entitou *Mockup*, ktoré obsahujú elementy (entita *Element* a entita *MockupElement*, pomocou ktorých si môže používateľ vytvoriť návrh konkrétnej podstránky. Používateľ má možnosť vytváraním oblastí záujmu a spájaním mockupov vytvoriť klikateľný prototyp (entita *Prototype*), ktorý je následne možné priradiť jednotlivým úlohám (taskom). Každá úloha má stanovenú svoju ukončovaciu podmienku, zväčša viazanú na konkrétny element. V rámci projektu si môže používateľ nahrávať rôzne obrázky/screenshoty (entita *Image*), ktoré sú používané ako pozade jednotlivých elementov v podstránkach prototypu.

Počas vykonávania jednotlivých úloh je zaznamenávaná rozličná aktivita používateľa, ktorá sa ukladá v entite *Record*. Následne sú zaznamenané dáta spracované a výsledky sú uložené v entite *Result*.

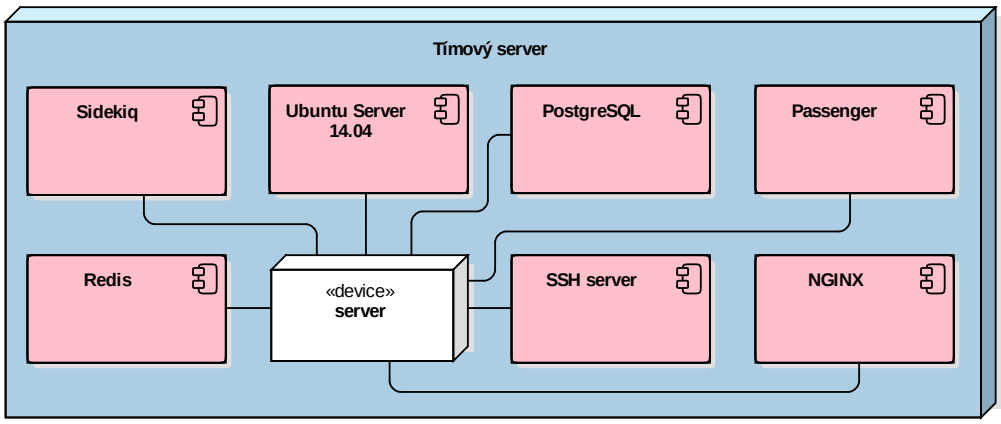


Obr. 18: Diagram dátového modelu.

2.3.5 Tímový server

Prevádzku našej tímovej stránky, ako aj nášho produktu, zabezpečuje tímový server. Ako je možné vidieť na obrázku č. 19, server funguje na operačnom systéme Ubuntu Server 14.04. Beží na ňom webové server NGINX, ktorý obsluhuje statickú tímovú stránku, ako aj frontend aplikáciu.

Pre náš RoR backend server, beží pod NGINX aplikačný server Passenger, ktorý zabezpečuje spustenie RoR aplikácie. Ako databázu pre našu backend aplikáciu sme si vybrali PostgreSQL. Je tiež potrebný spustený Sidekiq, ktorý ďalej vyžaduje Redis. Sidekiq je potrebný pre vyhodnocovanie výsledkov testov.



Obr. 19: Diagram komponentov znázorňujúci jednotlivé moduly systému.

Pre prístup k serveru využívame SSH pripojenie. To zabezpečuje prístup nie len pre jednotlivých členov tímu, ale aj pre nástroje kontinuálnej integrácie, pri procese nasadzovania novej verzie nášho produktu na server.

Prevádzka viacerých aplikácií vrámci jedného doménového priestoru. Počas nastavovania servera sa vyskytli komplikácie s tým, že pre nás nie je možné využívať subdomény a preto spustenie dvoch (troch) rozdielnych aplikácií na našom serveri muselo byť vyriešené cez suburi. Bolo preto potrebné nastaviť server tak, aby správne smeroval všetky requesty. Backend nastavenie bolo možné vyriešiť prostredníctvom nastavenia parametra pre Passenger aplikačný server. Dodatočná konfigurácia bola potrebná aj pre frontend, kde sa musí zasahovať aj do index.html, aby linky na stránke boli smerované na správne adresy.

2.4 Moduly systému

2.4.1 Nahrávanie obrázkov

Analýza Súčasťou testovania použiteľnosti webových stránok bude aj možnosť vytvoriť testy, kde bude použitý screenshot webovej stránky. Tvorca testu bude teda potrebovať spôsob, ako tento screenshot (obrázok) nahrať do editora, aby ho mohol použiť.

Návrh Pre realizáciu tejto funkcionality je potrebné vytvoriť používateľské rozhranie do frontend aplikácie, ako aj API do backend-u. API musí umožňovať tieto funkcie:

- nahranie nového obrázku,
- získanie konkrétneho obrázku
- získanie všetkých obrázkov patriacich k určitému projektu.

Získavanie obrázkov musí umožniť vrátiť obrázky v rôznom rozlíšení (napríklad menšie rozlíšenie sa použije vtedy, keď zobrazujeme viacero náhľadov na obrázky, ktoré používateľ nahral).

Implementácia Ako knižnicu, ktorá bude zabezpečovať ukladanie obrázkov na strane backend-u sme vybrali Dragonfly[1]. Tá umožňuje jednoduché priradenie obrázku do modelu cez dragonfly_accessor. Rovnako aj umožňuje získavanie nahraných obrázkov v rôznych rozlíšeniach pričom tieto obrázky generuje on-the-fly cez ImageMagick. V odpovedi na GET obrázku je potom URL adresa priamo na obrázok v požadovanom rozlíšení, rozlíšenie sa špecifikuje ako parameter resolution v GET požiadavke – ImageMagick formát[2].

Na frontend-e sa pre nahrávanie používa knižnica angular-file-upload[3]. Tá FileUpload triedu, ktorej inštancia vykonáva upload na určitú URL. Umožňuje obmedziť upload iba na určitý formát súborov, takisto umožňuje získanie informácie o progrese upload-u. Umožňuje nahranie viacerých obrázkov naraz – obrázky sú nahrávané postupne, nie asynchrónne všetky naraz.

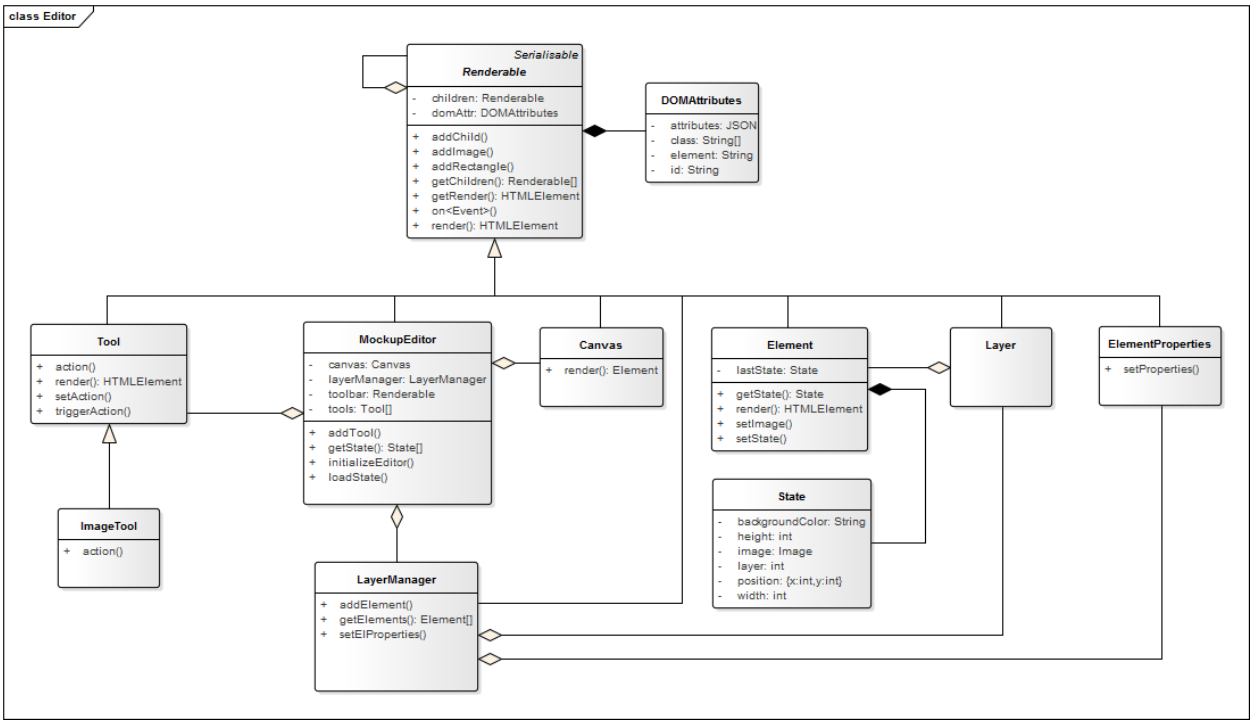
Testovanie Pre API boli vytvorené rspec testy, ktoré testujú kontrolér a model pre obrázky (Image). Počas testovania frontend-u sme prišli na to, že FileUpload obchádza Devise token-y a API už následne nevie autentifikovať používateľa. Bolo teda potrebné dorobiť túto funkcionalitu.

2.4.2 Mockup – Editor

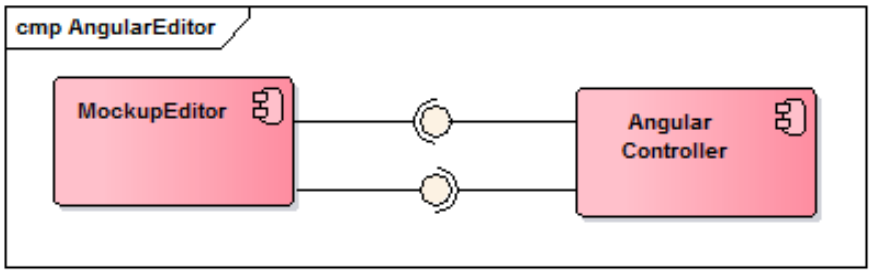
Mockup-Editor, ktorého diagram je znázornený na obr. č. 20 je jedným z komponentov načrtnutých v kapitole 2.3.3. Tento komponent slúži na vytváranie mockupov, ktoré sú používané pri testovaní.
Mockup-Editorje postavený na triede *Renderable*, ktorá obsahuje metódu *render*. Táto metóda po zavolaní vráti príslušný html element k danej triede, ktorý je vytvorený na základe objektu *DOMAttributes*. *Renderable* v sebe agreguje ďalšie pole typu *Renderable*. Táto stromová štruktúra sa využíva práve pri volaní render, kde výsledný dom element v sebe obsahuje aj elementy detských *Renderable*. Ďalšími zaujímavými metódami, sú metódy zo skupiny *on<Event >*. Takéto metódy sú po zavolaní *render*, naviazané ako callbacky dom elementu, ktoré sú volané prehliadačom pri spustení príslušného eventu.

Trieda *MockupEditor* dedí od triedy *Renderable*. Predstavuje mockup editor, ktorý je zobrazovaný používateľovi. Keďže táto trieda v sebe agreguje aj ostatné triedy potrebné na funkcionalitu mockupu. Z pohľadu anguláru, komunikácia vyzerá tak ako je znázornená na obrázku č. 21.

[1] `http://markevans.github.io/dragonfly/`
[2] `http://markevans.github.io/dragonfly/imagemagick/`
[3] `https://github.com/nervgh/angular-file-upload`



Obr. 20: Diagram tried editora.



Obr. 21: Vzťah medzi angularjs a editorom.

MockupEditor poskytuje rozhranie pre angular a to konkrétne metódy *getState* a *loadState*, ktoré slúžia na perzistenciu editora, resp. elementov mockupu. Angular na druhej strane poskytuje editoru metódy, ktoré slúžia na upload obrázka pre element. Takéto oddelenie logiky editora od angularu nám umožňuje ľahké prepínanie medzi editormi v rámci kontroleru a keďže dedí od triedy *Renderable*, tak aj jeho následne vykreslenie.

MockupEditor v sebe agreguje ostatné triedy, ktoré zabezpečujú funkcionalitu editora ako napr. *Tool*, *Canvas* alebo *Element*. Tieto triedy sú bližšie opísané nižšie.

Tool *Tool* je takisto *Renderable* trieda. Je súčasťou toolbaru, ktorý vykresluje editor. HTML Element, ktorý *Tool* predstavuje, je pomocou jquery-ui knižnice nastavený ako Draggable. HTML Element má k sebe pomocou jquery priradený svoj prislúchajúci *Tool* objekt. Každý *Tool* so sebou nesie akciu, ktorú využíva *Canvas*. *Tool* je pri spustení akcie zodpovedný za vytvorenie vhodného *Element*-u, ktorý bude vložený do *Canvas*-u. Pre zabezpečenie základnej funkcionality sme vytvorili *ImageTool*, ktorý pri jeho pustení do *Canvas*-u umožní používateľovi vybrať obrázok (screenshot) a pridá ho do *Canvas*-u. Pre zobrazenie dialógu

na výber obrázka je použité už spomínané angular rozhranie.

Canvas Tvorí plochu, v ktorej sa mockup vytvára. Obsahuje objekty triedy *Element*. Všetky elementy v *Canvas*-e sú vďaka jquery-ui Draggable a Resizable a je ich možné ľubovoľne presúvať a meniť ich rozmer, ktorý sa následne uloží do ich stavu. Samotný *Canvas* je (opäť z jquery-ui) Droppable, je možné nad ním pustiť niektorý z *Tool*-ov. Týmto sa spustí akcia priradená konkrétnemu *Tool*-u, ktorá vráti novovytvorený *Element*, ktorý si *Canvas* do seba uloží.

Element Trieda *Element* predstavuje element mockupu, napríklad button alebo obrázok navigačného menu. Pole týchto tried predstavuje stav celého mockupu. Stav daného *Element*-u je uložený v triede State, ktorá sa následne ukladá pomocou API.

LayerManager a Layer Trieda *LayerManager* je *Renderable* trieda, ktorá predstavuje panel s elementami v mockupe. Okrem zobrazovania zoznamu elementov sa táto trieda stará aj o ich prekrývanie pomocou *z-index*-ov, kde poradie elementov v tomto zozname určuje aj ich reálne prekrývanie. Trieda *Layer* je reprezentácia konkrétneho *Element*-u v triede *LayerManager*.

ElementProperties Táto trieda zabezpečuje zobrazenie informácií o práve vybranom elemente.

2.4.3 Zaznamenávanie aktivity používateľov

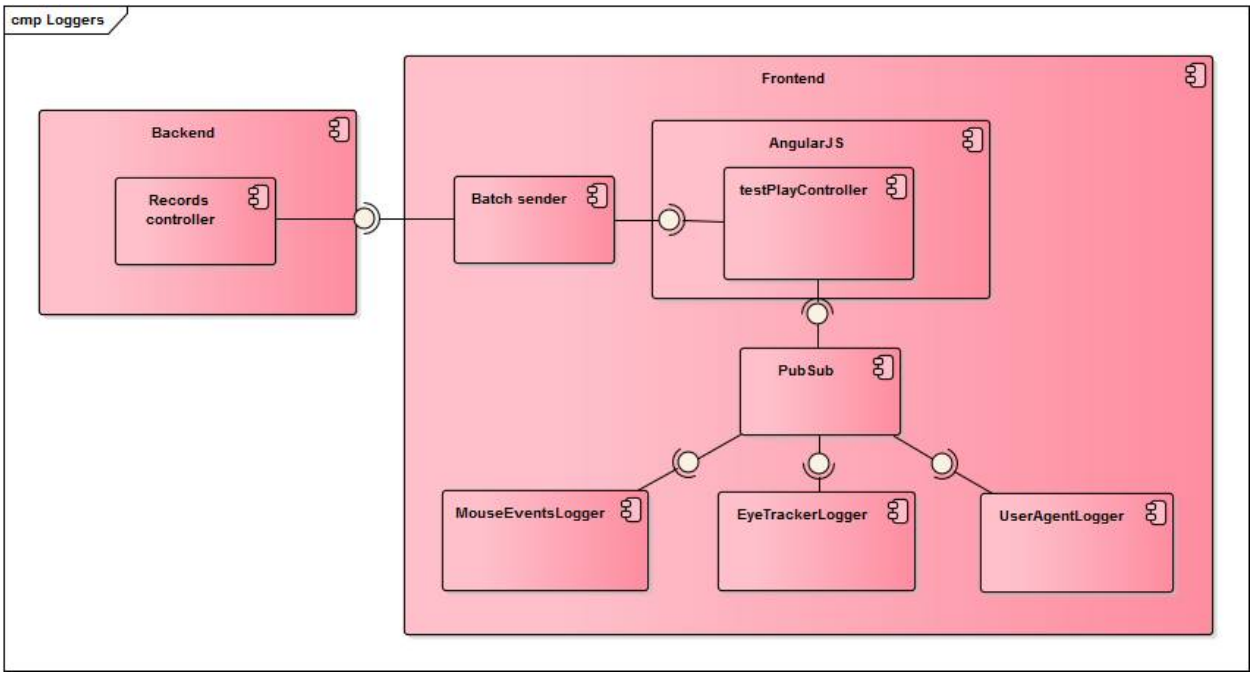
Analýza Veľmi podstatnou súčasťou nášho riešenia je zaznamenávanie aktivity používateľov počas vykonávania úloh, aby sme následne boli schopní nad týmito zaznamenanými dátami vykonávať analýzu a vyhodnocovať uskutočnené testy. V rámci nášho riešenia zaznamenávame nasledovné údaje od používateľov:

- pohyb myši
- kliknutia myši
- scrollovanie
- pohľad používateľa
- údaje o používateľovi (prehliadač, operačný systém a rozlíšenie obrazovky)

Návrh a implementácia Pre zaznamenávanie aktivity používateľov sme vytvorili niekoľko loggerov, ktoré sú špecializované na zaznamenávanie určitého typu aktivity používateľov. Implementované loggery sú:

- *MouseEventsLogger* - zaznamenávanie pohybu myši, kliknutí a scrollovania
- *EyeTrackerLogger* - zaznamenávanie pohľadu používateľov s využitím UX infraštruktúry dostupnej u nás na fakulte, pričom logger sa dopytuje k sledovaču pohľadu pomocou API rozhrania a tak získava údaje o aktuálnej pozícii pohľadu používateľa.
- *UserAgentLogger* - zaznamenávanie údajov o používateľovi. Konkrétne sa zaznamenáva typ operačného systému, typ webového prehliadača a rozlíšenie obrazovky.

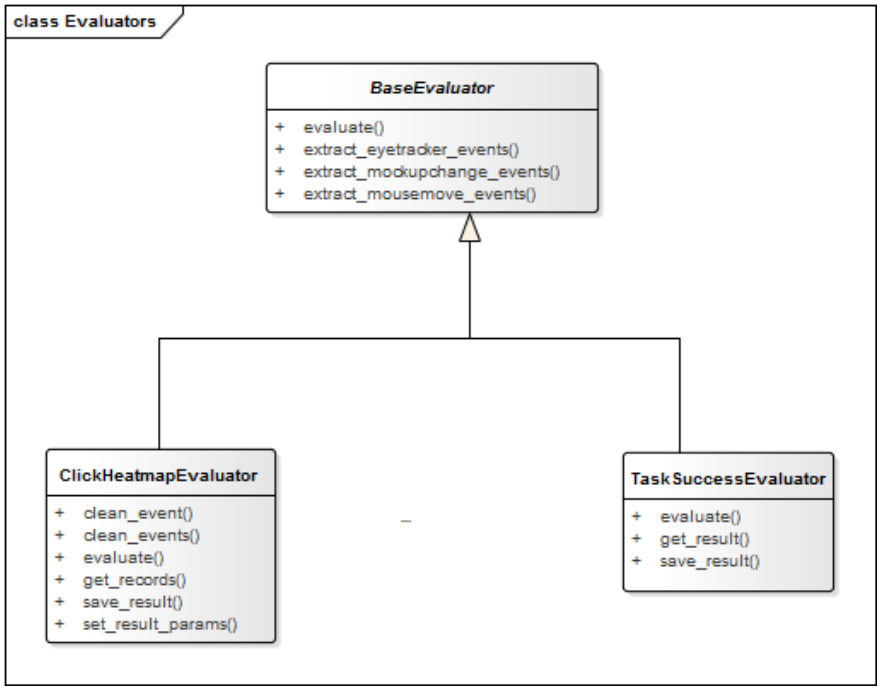
Vytvorené loggery komunikujú s frontendom (AngularJS) pomocou knižnice *PubSub*, ktorá reprezentuje návrhový vzor publish-subsribe, pričom loggery publikujú zaznamenané dáta (publish) a frontendový kontroler *testPlayController* príjma tieto dáta (subsribe) a následne tieto dáta pomocou knižnice *BatchSender* posiela na backendový kontroler *RecordsCotroller*, ktorý tieto dáta ukladá do databázy. Rozloženie a vzájomná komunikácia týchto komponentov je znázornená na Obrázku č. 22.



Obr. 22: Vzťah medzi komponentmi umožňujúcimi zaznamenávanie aktivity používateľov.

2.4.4 Vyhodnocovanie testov a práca so záznamami

Po tom čo používateľ dokončí testovanie príslušný kontroler (*TaskRunController* alebo *TestRunController*) pomocou *sidekiq*-u inicializuje vyhodnocovanie záznamov. Toto vyhodnocovanie zabezpečujú triedy z hierarchie, ktorá je znázornená na obrázku č. 23.

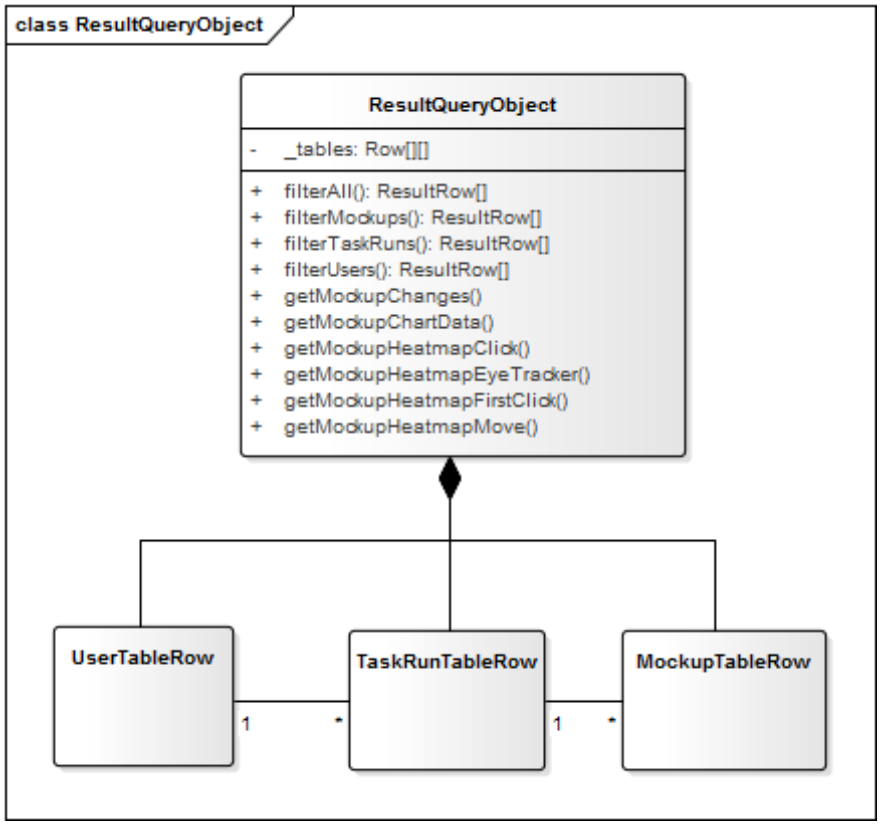


Obr. 23: Hierarchia tried vyhodnocovania.

Abstraktná trieda *BaseEvaluator* predstavuje rozhranie a poskytuje pomocné funkcie pre konkrétne vyhodnocovače. Triedy ako *ClickHeatMapEvaluator* alebo *TaskSuccessEvaluator* okrem iných potom zabezpečujú vyhodnotenie výsledkov z konkrétneho hľadiska napr. vytvoria dáta potrebné pre zobrazovanie heatmapy klikov alebo pohľadu používateľa.

Tieto výsledky na frontende zabaľuje trieda *ResultQueryObject* znázornená na obr. č. 24. Tento objekt je vytvorený spracovaním výsledkov z backend API. Tvoria ho tri tabuľky, ktoré na seba odkazujú cudzím kľúčom a predstavujú výsledky, ktoré môžu byť jednak viazané na celý *Task* (napr. dĺžka jeho trvania, úspešnosť) alebo na konkrétny mockup (kliky, pohľady očí).

Filter metódy tohto objektu, potom zabezpečujú filtrovanie a spájanie týchto tabuliek. Paralelou ich funkcionality môžu byť SQL dopyty a *JOIN*. Takýmto spôsobom vieme zabezpečiť funkcionalitu keď chceme napríklad kliky pre jeden mockup od všetkých používateľov, ktorým sa nepodarilo dokončiť úlohu. *get* metódy tohto objektu potom používaju filtre a zjednodušujú prístup k výsledkom a ich formátovanie potrebné pre grafy a podobne.



Obr. 24: Trieda *ResultQueryObject*.

A Zoznam kompetencií tímu

Peter Dubec Už pred štúdiom na FIIT som mal veľký záujem o webové technológie, takže som sa naučil základy HTML a CSS. Počas štúdia som všetky zadania, pokiaľ to bolo možné, vypracovával v jazyku Java. Následne po absolvovaní Výskumne orientovaného semináru, kde som sa zoznámil s webovým frameworkom Ruby on Rails ma tento framework veľmi zaujal a počas leta som absolvoval letnú školu Startup Summer School, kde som si zdokonalil svoje znalosti ohľadom tohto frameworku a taktiež si vyskúšal prácu s rôznymi ďalšími zaujímavými technológiami, ako napríklad Git, či rôznymi databázovými systémami ako Postgres, ElasticSearch či Redis. V rámci svojej bakalárskej práce som pracoval na vytvorení automatického klasifikátoru dokumentárnych filmov, ktorý som vyvíjal v jazyku Ruby a taktiež v rámci mojej bakalárskej práce bolo potrebné vytvoriť niekoľko parserov, ktoré som taktiež vyvíjal v jazyku Ruby. V rámci overenia mojej bakalárskej práce som uskutočnil hromadný experiment, pri ktorom bolo využité sledovanie pohľadu účastníkov, vďaka čomu som mal možnosť vyskúšať si prácu s technológiou umožňujúcou sledovanie pohľadu. V rámci mojej diplomovej práce budem pracovať na automatizácií používateľských študií s využitím sledovania pohľadu.

Róbert Cuprik S webovými technológiami som sa stretol prvýkrát na strednej škole, kde som robil web stránky pomocou PHP. Následne som sa neskôr venoval najmä HTML, CSS a javascriptu. Počas štúdia na FIIT som programoval v jazyku C a Java. S Ruby on Rails som začal pracovať na predmete VOS, kde som implementoval stránku na kolaboratívne učenie, ktorá využívala aj websockety. Popri škole som sa venoval aj javascriptu a webgl, kde som programoval vlastný 3D engine pre hru v prehliadači. Témou mojej bakalárskej práce bolo hľadanie motívov v DNA sekvenciách, jej výsledky som prezentoval na IIT.SRC. Tejto oblasti sa venujem aj pri DP.

Patrik Gajdošík Počas štúdia na fakulte som sa zameriaval hlavne na vývoj v programovacom jazyku Ruby. Všetky zadania a rovnako aj výsledok mojej bakalárskej práce (ktorá bola prezentovaná aj na IIT.SRC 2015) boli v prípade možnosti písané v tomto jazyku. Zaujímam sa rovnako aj o systémové programovanie v jazyku C. Mám dlhoročné skúsenosti s prácou v operačnom systéme GNU/Linux.

Roman Roba Primárne sa zaujímam sa o vývoj aplikácií pre operačné systémy Mac-OS X a iOS v jazyku Obj-c. Tieto skúsenosti uplatňujem prímarne pracovne, no uplatnil som ich aj pri práci na bakalárskom projekte. Ďalej sa zaujímam o .NET-ový framework a jazyk C#, kde mám rovnako pracovné skúsenosti. Školské zadania som riešil ako v Obj-c, tak aj C#. V budúcnosti sa plánujem zamerať na vývoj aplikácií v jazyku Swift.

Monika Sanyová V práci počas štúdia som ako backend vývojárka nadobudla skúsenosti s prácou vo frameworku Ruby on Rails. Taktiež mám základné skúsenosti s programovaním v Javascripte, či s jazykmi HTML a CSS. Zaujímam sa o webové technológie, preto beriem ako príležitosť naučiť sa nové technológie, akými sú napríklad framework Angular, či preprocesor SASS. V rámci diplomovej práce budem využívať Eyetracker technológiu na sledovanie pohľadu pri testovaní UX, takisto ako v našom tímovom projekte. V bakalárskej práci som vyvíjala mobilnú aplikáciu vo frameworku Android, ktorý je pre mňa taktiež zaujímavou technológiou. Školské projekty som implementovala prevažne v jazyku Java, pričom v poslednom roku štúdia som preferovala framework Ruby on Rails. Počas štúdia som sa oboznámila s prácou s verziovacím nástrojom Git a vyskušala som si prácu s databázami MySQL, PostreSQL.

Jakub Vrba Počas bakalárskeho štúdia som vypracovával projekty prevažne v jazyku Java, v ktorom som implementoval aj bakalársku prácu zameranú na spracovanie obrazu. Na predmete Výskumne orientovaný seminár som sa zoznámil s frameworkom Rails a začal sa viacej zaujímať o webové technológie. V práci som sa venoval frameworku .NET a pracoval som aj na frontende s Javascriptom, HTML a CSS. Čo sa týka

Zoznam kompetencií tímu pokračuje na dalšej strane...

databázových technológií mám skúsenosti s PostgreSQL a Oracle. V súčasnosti pracujem na backende v Jave a na frontende v AngularJS. V diplomovej práci sa zaoberám spracovaním nahrávok z testov UX.

Tomáš Žigo V priebehu štúdia na FIIT som sa venoval hlavne programovaniu v jazykoch Java a C, v ktorom som robil aj svoju bakalársku prácu zameranú na paralelné výpočty na grafickej karte. Pri práci s databázami som sa stretol hlavne s databázou PostgreSQL, ktorú som používal na viacerých projektoch. Ku koncu štúdia som sa venoval webom – HTML, CSS a Javascript - na predmete Webové publikovanie. Zaujímam sa hlavne o frontend vývoj v javascripte.

Róbert Cuprik	Rails, Javascript, HTML, CSS, PostgreSQL
Jakub Vrba	AngularJS, Java, Rails, Javascript
Peter Dubec	Ruby, Java, HTML, CSS, PostgreSQL
Roman Roba	iOS, Objective-C, C#, .NET
Patrik Gajdošík	C, Ruby, Rails, GNU/Linux
Monika Sanyová	Rails, HTML, CSS, Andriod, Java
Tomáš Žigo	HTML, Java, C, PostgreSQL

Tabuľka 3: Zoznam kompetencií členov tímu

B Metodiky

M01 - Kontinuálna integrácia

Kontinuálna integrácia (Continuous Integration, CI).

Obsah

1. Wercker
2. Testovanie
3. Nasadenie
4. Závislosti
5. Prístup k serveru

Wercker

Na kontinuálnu integráciu sa používa Wercker. Každá jedna zostava (build) a aj nasadzovanie prebieha v docker kontajneri. Wercker postupne vykonáva kroky, ktoré sa definujú konfiguračnom súbore `wercker.yml`:

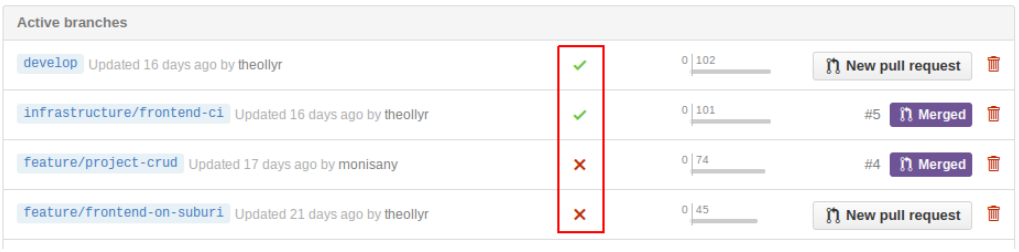
```
# Docker kontajner, v ktorom sa uskutočnia všetky kroky.
box: mwallasch/docker-ruby-node
# Toto je build pipeline. V nej prebiehajú všetky kroky počas testovania.
# Viac sa môžete dozviedieť tu:
# http://devcenter.wercker.com/docs/pipelines/index.html
# http://devcenter.wercker.com/docs/steps/index.html
build:
    steps:
        # Každý krok môže mať definované ešte ďalšie svoje nastavenia.
        - bundle-install:
            without: development production deployment
        - script:
            name: rspec
            code: bundle exec rspec

# Toto je deploy pipeline. Spúšťa sa v prípade spustenia nasadenia.
deploy:
    # codereview vymedzuje kroky špecifické iba pre určitý druh nasadenia.
    codereview:
        - script:
            name: install rubycritic
            code: gem install rubycritic --no-rdoc --no-ri
    # Kroky, ktoré sa vykonajú po vykonaní všetkých krokov v pipeline.
    # Sú vykonané bez ohľadu na to, či niektorý z predchádzajúcich krokov
    # zlyhal alebo nie.
    after-steps:
        - theollyr/slack-notifier
```

Ak ktorýkoľvek krok zlyhá, zvyšné kroky (okrem `after-steps`) sa už nevykonajú.

GitHub Integrácia

Wercker sa automaticky integruje do rozhrania GitHub-u, sú tam zobrazené stavy jednotlivých zostáv v závislosti od vetvy. V prípade, že zostava vetvy zlyháva, je potrebné opraviť chyby, ktoré spôsobujú neúspešné testy.



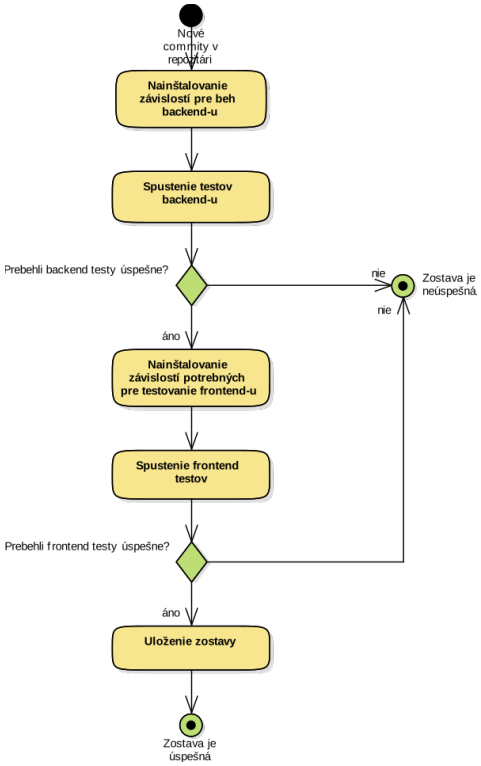
Testovanie

Testovanie prebieha v dvoch častiach:

1. testovanie backendu - spúšťajú sa `rspec` testy;
2. testovanie frontendu - spúšťajú sa `jasmine` testy.

Ak niektorý test zlyhá, zvyšok testovania sa nevykoná a zostava (angl. build) je označená za neúspešnú.

Proces testovania prebieha v docker kontajneri a spúšťa ho služba Wercker. Su vykonávané viaceré kroky. Ako je možné vidieť na obrázku, po tom, ako sa zaznamenajú nové zmeny v zdrojových kódoch, sú tieto kódy stiahnuté. Pre beh je potrebné taktiež nainštalovať závislosti a následne sa spúšťajú testy, najskôr backend a potom frontend.



> Ak nechceme, nie je potrebné, aby sa push-nuté commity otestovali (napríklad došlo iba k zmene v README), tak hocikam do názvu commitu dáme `[skip ci]` alebo `[ci skip]`.

Neúspešná zostava

`develop` vetva by mala byť vyvíjaná tak, aby nikdy nenastal stav, kedy jej zostavenie nebude úspešné. CI testuje aj každý PR a preto pred jeho merge-nutím, je potrebné skontrolovať, či CI označila tento PR za úspešný. V prípade, že nie, je najprv potrebné opraviť chyby, ktoré spôsobujú zlyhanie zostavy.

Nasadenie

V prípade úspešného zbehnutia všetkých testov je možné takúto zostavu nasadiť na server. Všetky zmeny na vetve `develop` sú automaticky nasadené do staging-u.

Nasadenie z inej vetvy sa môže vykonať tiež po spoločnom odsúhlasení.

Na nasadenie sa využíva Capistrano a Grunt. Capistrano zabezpečuje nasadenie backendu, Grunt frontendu. Každý z týchto dvoch nástrojov definuje úlohy (task-y), ktoré sa vykonávajú.

Konfigurácia Capistrano-a sa nachádza v `Capfile`, `config/deploy.rb` a `config/deploy/production.rb`. Konfigurácia Grunt-u je v `Gruntfile.js`.

Závislosti

Závislosti, ktoré je potrebné nainštalovať, je potrebné rozdeliť tak, aby sa vo fáze testovania nainštalovali iba tie, ktoré sú potrebné pre testovanie a vo fáze nasadenia len tie, ktoré sú potrebné pre nasadenie.

Bundler

Počas testovania sa inštalujú len gem-y, v skupine `testing` (+ globálne závislosti). Po nasadení sa inštalujú len gem-y v skupine `production`.

Grunt

Počas testovania sa inštalujú len balíčky v súbore `package/development.json`. Po nasadení sa inštalujú len balíčky v `package/production.json`.

Pre manuálne nainštalovanie JS závislostí z iného ako základného `package.json` súboru, je možné použit príkaz:

```
$ grunt package:production
$ npm install
```

Dôjde k výmene niektorého z `{development,production,default}.json` súborov v `package` za `package.json` v koreni frontendu a `npm install` potom použije tento súbor pre inštalovanie závislostí.

Bower

Balíčky sa inštalujú počas testovania aj po nasadení.

Prístup k serveru

Na prístup k serveru sa používa SSH. Pre prístup k serveru je potrebné požiadať jeho administrátora o vytvorenie používateľského účtu. Ku každému účtu je potrebné prideliť SSH kľúč, ktorý sa bude používať na overenie.

Pre vytvorenie SSH kľúča:

```
$ ssh-keygen -t rsa
```

Skopírovanie verejného kľúča na účet na serveri:

```
$ ssh-copy-id <username>@147.175.149.143
```

M02 - Git

1. Naklonovanie celého projektu k sebe

Na github je https/ssh link na clone projektu, treba ho skopírovať a do konzoly dať príkaz:

```
git clone <ten skopírovaný link>
```

2. Vytvorenie novej branche z `develop` -u

```
git checkout develop
git pull origin develop
git checkout -b nazov-novej-branche
```

Teraz som už v novej branchi.

3. Naklonovanie existujucej branche z githubu

```
git fetch
git checkout -b name origin/name
```

4. Spôsob pomenovania branche

Ak je to nová feature -> `feature/kratky-popis-feature`

Ak je to fix starej feature -> `fix/kratky-popis-starej-feature`

5. Pull request

Pull request (PR) vytváram, keď mám funkčnú, dokončenú, otestovanú feature, ktorú chcem mergnuť s `develop`-om. Postup, keď som v mojej feature branchi a idem vytvárať PR:

```
git add .
git commit -m "zmysluplna sprava o poslednych zmenach v mojej feature branchi"
git checkout develop
git pull origin develop # teraz mám najnovšiu verziu developu
git checkout feature/moja-feature
git merge --no-ff develop
```

Ak nastali konflikty, vyriešim ich, uložím zmeny novým commitom, ktorého názov bude napr. `resolve merge conflicts` a dokončím merge znovu prikazom `git merge --no-ff develop`.

Ak nenastali konflikty, teraz mám vo svojej feature branchi mergnutý kód. Tento kód ešte raz otestujem, spravím codereview a až keď som si istý, že je to v poriadku, pushnem to na git:

```
git push origin feature/moja-feature
```

Priamo na github (uz nepoužívam konzolu) vytvorím PR na megnutie mojej feature branche a `develop`-u, kde pozvem všetkých, ktorí sú relevantní pre danú branch a počkám na to, kým reviewer nepozrie kód a spraví merge.

M03 - Jasmine

Unit testy

- Pomocou nich testujeme zvolené jednotky kódu, či fungujú tak, ako predpokladáme
- Používať TDD
- Používať angličtinu

Príklad

“Ako používateľ sa chcem prihlásiť a po prihlásení ma presmeruje na dashboard“

Túto story rozbijeme do testov nasledovne: story -> features -> units Feature napríklad môže byť prihlasovací formulár, test pre otestovanie by vyzeral nasledovne:});

```
describe('user login form', function() {
    // critical
    it('ensure invalid email addresses are caught', function() {});
    it('ensure valid email addresses pass validation', function() {});
    it('ensure submitting form changes path', function() { });

    // nice-to-haves
    it('ensure client-side helper shown for empty fields', function() { });
    it('ensure hitting enter on password field submits form', function() {
```

Konkrétny príklad testu:

```
describe("Unit Testing Examples", function() {
  beforeEach(angular.mock.module('App'));
  it('should have a LoginCtrl controller', function() {
    expect(App.LoginCtrl).toBeDefined();
  });
  it('should have a working LoginService service', inject(['LoginService',
    function(LoginService) {
      expect(LoginService.isValidEmail).not.to.equal(null);
      // test cases - testing for success
      var validEmails = [
        'test@test.com',
        'test@test.co.uk',
        'test734ltylytkliytkryety9ef@jb-fe.com'
      ];
      // test cases - testing for failure
      var invalidEmails = [ 'test@testcom',  'test@ test.co.uk', 'ghgf@fe.com.co.',
      'tes@t@test.com',  ' '];
      // you can loop through arrays of test cases like this
      for (var i in validEmails) {
        var valid = LoginService.isValidEmail(validEmails[i]);
        expect(valid).toBeTruthy();
      }
      for (var i in invalidEmails) {
        var valid = LoginService.isValidEmail(invalidEmails[i]);
        expect(valid).toBeFalsy();
      }
    }])
```

```
  );
});
```

Describe, it

Pre describe použiť názov featury, ktorá sa ide testovať a v it krátko popísať funkcionalitu testu, začať slovesom.

Užitočné odkazy:

- http://andyshora.com/unit-testing-best-practices-angularjs.html
- http://jasmine.github.io/

M04 - Jira

Všeobecné

- používať anglický jazyk
- všetky novo vytvorené *tasky* a *story* pridávať do backlogu
- nepridávať a neodoberať úlohy počas behu šprintu
- spravovať si stavy pridelených úloh
- na prihlásenie použiť svoje údaje zo systému AIS

Úlohy

- úlohy sa prideľujú vždy na začiatku šprintu
- náročnosť každej *user story* sa určuje spoločne
  - individuálnym hodnotením a následnou konzultáciou, kým celý tím nedôjde k rovnakému hodnoteniu
  - v pomere obtiažnosti k referenčnej *user story*
  - v prípade veľkej náročnosti sa vytvorí *epic* ktorý rozdeľuje *user story* naprieč viacerými šprintami
- ak je to možné, treba úlohe priradiť informačný popis vyjadrený popismy:
  - *feature*
  - *infrastructure*
  - *management*
  - *develop*
- pri náročnej úlohe si poverená osoba túto úlohu rozdelí na pod úlohy, ktoré následne môže prideliť niekomu ďalšiemu
- každý rieši iba aktuálne pridelené úlohy pre daný šprint
- nové úlohy sa vytvárajú vždy spoločne na stretnutí tímu
- pri nájdení chyby(bugu) treba túto úlohu pridať do systému
  - ako úlohu typu *bug*
  - treba si najprv overiť či daná chyba už v systéme nieje evidovaná
  - v prípade, že vieme určiť zodpovednú osobu, túto úlohu jej treba prideliť
- ak by nastala situácia, že v systéme existujú dve rovnaké úlohy, tieba úlohy s novším dátumom označiť v systéme za klon pôvodnej úlohy
- úloha sa považuje za ukončenú, až keď boli splnené všetky požiadvky kladené v danej úlohe
- úlohu je nutné uzatvoriť v momente jej splnenia tj. nečakať na koniec šprintu
- pri zatváraní úlohy, treba uviesť čas strávený riešením úlohy
- po vyriešení úlohy treba túto úlohu dať do stavu *resolved*
- do stavu *closed* sa úloha dostane vtedy, keď na tímovom stretnutí všetci odsúhlasia jej úplnosť

Šprint

- dĺžka trvania špintu sú 2 týždne(10 pracovných dní)
- plánovanie šprintu prebieha na pondelňajšom stretnutí, kedy sa ohodnotia nové úlohy. Následne sa na základe náročností a schopnosti týmu vyberie určitý počet úloh, ktoré sa budu v tomto šprinte riešiť
- oficiálny začiatok šprintu je po ukončení pondelňajšieho stretnutia tímu
- oficálny koniec šprintu je pred pondelňajším stretnutím tímu
- po ukončení sa šprint spoločne vyhodnocuje na základe úspešnosti dokončených úloh

JIRA

- odkaz na jiru

M05 - JavaScript

Užitočné odkazy:

- airbnb

Angular.js

Súborová štruktúra

Používať predpripravenú súborovú štruktúru:

```
/angular-app
  /controllers
  /directives
  /filters
  /modules
  /services
  /templates
```

Vytvárať podpriečinky tak, aby spájali logické celky dohromady, napríklad:

```
/angular-app
  /controllers
    /project
       projectController.js
       projectDetailController.js
  /directives
    /project
    /user
  /filters
  /modules
  /services
  /templates
```

Konvencie pomenovávania

Používať v názve súboru typ angulárovskeho objektu:

```
projectController.js
userLoginDirective.js
```

Používať camelCase pre js súbory:

```
projectController.js
```

Súbory by mali mať rovnaké meno ako angulárovský objekt projectController.js

```
suxessControllers.controller('ProjectController', [...]);
```

Názvy Controllerov sa začínajú s veľkým písmenom, názvy direktív s malým:

```
suxessControllers.controller('ProjectController', [...]);
suxessDirectives.directive('projectCreateDirective',[...]);
```

Pomenovať html súbory podľa ich prislúchajúceho angulárovského objektu, príklad:

```
suxessDirectives.directive('projectCreateDirective',[...]);
project-create-directive.html
```

SUXESS knižnica

Používať globálnu premennú SUXESS na definovanie triedy

```
SUXESS.SecureStateCheck = function () {...}
var secureCheck = new SUXESS.SecureStateCheck(..);
```

Nevytvárať nový namespace, príklad:

```
SUXESS.namespace.SecureStateCheck
```

Spájať triedy do priečinkov:

```
/suxess
    /utils
        secureStateCheck.js
    suxess.js
```

Štýl kódu

Používať pravidlá zadefinované v airbnb.

V stringoch používať ' namiesto ".

Pomenovanie by malo, čo najviac opisovať, či už triedu alebo metódu, podľa toho, čo ich vystihuje. Jediná výnimka sú premenné pri prechádzaní cyklu, tie môžu byť skrátene na jedno písmeno, začínajúc od písmena i.

```
variableNamesLikeThis
functionNamesLikeThis
ClassNamesLikeThis
methodNamesLikeThis
ConstantsLikeThis
```

Komentáre

Používať komentáre, ak je kód príliš komplexný a bol by viac zrozumiteľný s krátkym vysvetlením.

Príklad:

```
var x = 5; //variable x is set to 5, because...
/*
This is a
multi-line
```

```
comment
*/
```

Dokumentácia

Používať jsdoc na písanie dokumentácie pre metódy.

- Písať stručné správy.
- Vysvetliť funckionalitu metódy.
- Špecifikovať vstupné premenné, krátko vysvetliť premennú, ak treba.
- Špecifikovať návratovú hodnotu a typ návratovej hodnoty
- Písať dokumentáciu po anglicky

Príklad:

```
/**
 * Returns true if user is allowed to visit a given state.
 * @param state - from $stateProvider service
 * @param user
 * @returns {boolean}
 */
```

M06 - Ruby on Rails

Všeobecné

- používať odsadenie o veľkosti 2 medzier (nastaviť veľkosť tab)
- dĺžka riadku nesmie presahovať 80 znakov
- využívať anglický jazyk
- názvy tried, metód, premenných atď. musia byť rozumné a opisné
- naming: mená tried Camel case (napr. ClassName), všetko ostatné Snake case (napr. some_function_name), pričom konštanty všetko veľké písmená (SOME_CONSTANT)

- zátvorky nie sú nutnosťou, ale pre krajší kód je dobré ich uvádzať. Zátvorky neuvádzať, pokiaľ by nemali žiaden obsah, alebo je obsah príliš krátky

```
# Don't do this
def split_message()
  @msg.split() if (@msg.size > 20)
end

# Do this
def split_message
  @msg.split if (@msg.size > 20)
end
```

- nepoužívať zápornú podmienku pri `if` ale radšej použiť `unless`

```
# Don't do this
if !@read_only
end

# Use unless statement
unless @read_only
end
```

- nepoužívať zápornú podmienku pri `while` ale radšej použiť `until`

```
# Don't do this
while !@server.online?
end

# Use until statement
until @server.online?
end
```

- názvy metód, ktoré sú predikátom (tj. vracajú true/false) ukončovať otáznikom (napr. `admin?`)
- nepoužívať `for` ale `each` (for nevytvára lokálny scope)

```
# Don't do this
for color in colors
  puts color
end
```

```ruby
# Use each block
colors.each do |color|
  puts color
end
```

MVC

- využívajte konvencie MVC
- nemiešajte logiku (žiadna logika modelu v kontroleri a pod.)
- pokúšajte sa dodržať "Fat model, skinny controllers"

Kontroler

- udržujte ich jednoduché a čisté
- dodržujte REST metodiku (jeden kontroler = jeden Resource tj. kontroler User opisuje iba usera!)

Model

- nepoužívajte zbytočné skratky pre názvy modelov (napríklad nie `UsrHstr` ale `UserHistory`)
- neopakujte seba a ani nereplikujte funkcionalitu Rails/Ruby
- pokiaľ využivate rovnaku podmienku vo `find` viac krát, použite `named_scope`

```ruby
class User < ActiveRecord::Base
  scope :active, -> { where(active: true) }
  scope :inactive, -> { where(active: false) }

  scope :with_orders, -> { joins(:orders).select('distinct(users.id)') }
end
```

- využívajte "sexy" validácie

```ruby
# bad
validates_presence_of :email
validates_length_of :email, maximum: 100

# good
validates :email, presence: true, length: { maximum: 100 }
```

- vyhnite sa interpolácii stringov pri dopytoch

```ruby
# bad - param will be interpolated unescaped
Client.where("orders_count = #{params[:orders]}")

# good - param will be properly escaped
Client.where('orders_count = ?', params[:orders])
```

- pokiaľ chcete získať jeden záznam podľa id, preferujte `find` a nie `where`

```ruby
# bad
User.where(id: id).take

# good
User.find(id)
```

- taktiež preferujte `find` aj pri hľadaní podľa názvu atribútu

```ruby
# bad
```

```ruby
User.where(first_name: 'Bruce', last_name: 'Wayne').first

# good
User.find_by(first_name: 'Bruce', last_name: 'Wayne')
```

- pokiaľ potrebujete spracovať viacero záznamov, preferujte `find_each`

```ruby
# bad - loads all the records at once
# This is very inefficient when the users table has thousands of rows.
User.all.each do |user|
  NewsMailer.weekly(user).deliver_now
end

# good - records are retrieved in batches
User.find_each do |user|
  NewsMailer.weekly(user).deliver_now
end
```

- pokiaľ píšete vlastné SQL a chcete zachovať peknú syntax odriadkovania, použite `heredocks` + `squish`

```ruby
User.find_by_sql(<<SQL.squish)
  SELECT
    users.id, accounts.plan
  FROM
    users
  INNER JOIN
    accounts
  ON
    accounts.user_id = users.id
  # further complexities...
SQL
```

Komentáre

- pri skopírovaní bloku kódu z internetu uveďte jeho zdroj v komentári
- medzi # a komentárom umiestnite minimálne jednu medzeru

```ruby
# Comment
```

- umiestnite aspoň jednu medzeru medzi kódom a komentárom

```ruby
puts "Hi" # prints string
```

- dodržujte úroveň odsadenia, ktorú má kód

```ruby
def printf(str)
  # prints string
  puts "#{str}"
end
```

- zarovnávajte nasledujúce sa komentáre na konci riadku

```ruby
@variable         # variable comment
@longNameVariable # long name variable comment
```

Viacriadkové komentáre

- notáciu *begin/end* pre viacriadkové komentáre nepoužívajte

```
=begin
  Block
  of
  commented
  code
=end
```

- používajte # pre každý riadok komentárového bloku

```
  # Block
  # of
  # commented
  # code
```

Tvorba dokumentácie

- dokumentácia sa generuje pomocou nástroja YARD
- pre opis metód, tried a premenných používajte celé vety ukončené bodkou
- opisujte všetky dôležité(kľúčové) metódy
- vyjadrujte sa stručne a k veci
- pri dokumentovaní metódy zadefinujte v nasledovnom poradí
  - funkcionalitu
  - vstupné premenné
  - výstupnú premennú
- ak je to možné, špecifikujte aký dátový typ očakávate na vstupe
- ak je to možné, špecifikujte aký dátový typ bude na výstupe
- ak je zadefinovaný vstup/výstup, tak umiesntite jeden prázdny komentárový riadok za opisom funkcionality metódy
- dátové typy vždy uvádzajte vo vnútri hranatých zátvoriek

Example

```ruby
# Class for all evil Monsters.
class Monster

  # Init with name of the Monster.
  #
  # @param name [String] Monster name.
  def initialize(name)
    @name = name.capitalize
  end

  # @return [String] Returning name.
  def getName
    var_name = @name
    return var_name
  end

  # Reveal true Monster identity.
  def reveal
    puts "The Call of #{getName}!"
  end

end
```

YARD

- inštalácia

```
> gem install yard
```

- generuj dokumentáciu iba pre zdrojové súbory, kde nastala zmena

```
> yard doc monsterClass.rb
```

M07 - RSpec

Prečo testovať?

- menej chýb v kóde
- rýchlejšie a jednoduchšie manuálne pretestovanie
- jednoduchšie doimplementovanie novej funkcionality

Čo testovať v modeloch?

- validnú factory

```
it "has a valid factory" do
  expect(build(:factory_you_built)).to be_valid
end
```

- validácie

```
it { expect(developer).to validate_presence_of(:favorite_coffee) }
it { expect(meal).to validate_numericality_of(:price) }
```

- scopes

```
it ".visible return all visible projects" do
  undeleted_project = FactoryGirl.create :project
  deleted_project = FactoryGirl.create :deleted_project
  all_visible_projects = Project.all.visible
  expect(all_visible_projects).not_to include(deleted_project)
end
```

- vzťahy has_many, belongs_to a pod.

```
it { expect(classroom).to have_many(:students) }
it { expect(profile).to belong_to(:user) }
```

- callbacks (napr. before_action, after_create)

```
it { expect(user).to callback(:send_welcome_email).after(:create) }
```

- všetky metódy danej triedy

```
describe "public instance methods" do

  context "responds to its methods" do

    it { expect(factory_instance).to respond_to(:public_method_name) }

  end

  context "executes methods correctly" do
```

```
    context "#method_name" do

      it "does what it's supposed to..." do
        expect(factory_instance.method_to_test).to eq(value_you_expect)
      end

    end

  end

end

describe "public class methods" do

  context "responds to its methods" do

    it { expect(factory_instance).to respond_to(:public_method_name) }

  end

  context "executes methods correctly" do

    context "self.method name" do

      it "does what it's supposed to..." do
          expect(factory_instance.method_to_test).to eq(value_you_expect)
      end

    end

  end

end
```

Čo testovať v kontroleroch?

- response

```
it "returns JSON of the project just updated" do
  expect(json[:name]).to eql @project_attributes[:name]
  expect(json[:description]).to eql @project_attributes[:description]
end
```

- status

```
it { is_expected.to respond_with 201 }
```

Vytváranie inštancie pomocou `let`

```
# BAD
before { @resource = FactoryGirl.create :device }

# GOOD
let(:resource) { FactoryGirl.create :device }
```

Bloky `before` a `after`

vykonávané sú v poradí:

```
before :suite
before :context
before :example
```

```
after  :example
after  :context
after  :suite
```

...pričom:

```
before(:example) # vykoná sa pred každým unit testom
before(:context) # vykoná sa raz pred celou skupinou unit testou vrámci describe/context bloku, v ktorej je
```

Využitie `subject`

- ak viackrát v testoch testujem ten istý objekt
- v kontrolery je response z requestu ako default subjekt
- príklad použitia

```
# BAD
it { expect(assigns('message')).to match /it was born in Belville/ }

# GOOD -> možnosť využitia syntaxe is_expected.to
subject { assigns('message') }
it { is_expected.to match /it was born in Billville/ }
```

Describe, context, it

```
RSpec.describe "something" do
  context "in one context" do
    it "does one thing" do
    end
  end
  context "in another context" do
    it "does another thing" do
    end
  end
end
```

- pre `describe` použiť krátke a výstižné popisy metódy

```
describe '.class_method_name' do
  # code block
end

describe '#instance_method_name' do
  # code block
end
```

- pre `it` použiť krátky popis začínajúci slovesom (dlhý popis evokuje potrebu použitia `context`)

FactoryGirl + FFaker

```
FactoryGirl.define do

  factory :project do
    name FFaker::Lorem.word
    description FFaker::Lorem.sentence

    factory :deleted_project, class: Project do
      deleted true
    end

  end
```

```
end
```

použitie takejto factory:

```
FactoryGirl.create :project

FactoryGirl.create :deleted_project
```

Užitočné zdroje

- kostra testov pre model
- konvencie novej syntaxe v RSpec
- before/after bloky
- ffaker cheatsheet
- tutoriál na testovanie API
- tutoriál na factory

Code-review.md - Grip

M08 - Code Review

Prehliadky kódu

Prehliadky kódu sú povinné pri pull requestoch. Podľa metodiky M02 - Git , sa po zadaní pull requestu prizvú relevantní ľudia na prehliadku kódu. Automatické prehliadky kódu zabezpečuje server resp. CI. Backend kód je prezeraný pomocou ruby gemu rubycritic. Jeho výstupom je html stránka na ktorú je generovaný odkaz, ktorý sa nachádza na slacku v kanáli #codereview. Frontend je prezeraný jshintom.

Požiadavky na backend

Každý reviewer je povinný pri prezeraní kódu prezrieť aj výstup z rubycritic. Kód v pull requeste by mal mať známku A. Ak je tomu inak, autor je povinný zdôvodniť prečo sa táto známka nedá dosiahnuť. Ruby kód musí byť v súlade s ostatnými relevantnými metodikami - RoR metodika, RSpec metodika.

Požiadavky na frontend

Frontent javascript kód musí prejsť jshint testom bez errorov. Kód musí spĺňať požiadavky ostatných metodík - JS Metodika, Jasmine metodika. Výstup jshintu je dostupný na gite pri branchi pri každom pushnutí kódu.

C Export evidencie úloh

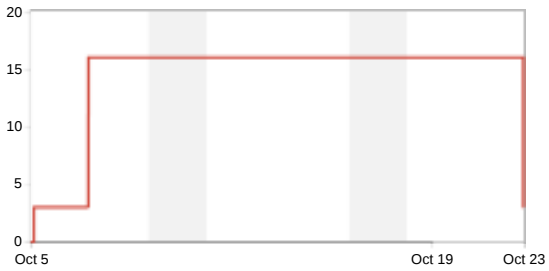
Export evidencie úloh pokračuje na dalšej strane...

sUXess

Sprint Report

Sprint 1

Closed Sprint, ended by Roman Roba 05/Oct/15 9:39 PM - 23/Oct/15 1:55 AM Linked pages View Sprint 1 in Issue Navigator



Status Report

* Issue added to sprint after start time

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (8)
UXWEB-2 *	Allow user to log in	Story	Major	CLOSED	8
UXWEB-3 *	High fidelity dashboard prototyp	Story	Major	CLOSED	-
UXWEB-7 *	Git methodics	Task	Major	CLOSED	-
UXWEB-8 *	Create a Git repo	Task	Major	CLOSED	-
UXWEB-9 *	Make a web page	Task	Major	CLOSED	-
UXWEB-10 *	Create basic structure for Rails API	Task	Critical	CLOSED	-
UXWEB-11 *	Create a basic structure for Angular	Task	Critical	CLOSED	-
UXWEB-20 *	Create a records for meeting 1, 2, 3 and 4	Task	Major	CLOSED	-

Issues Not Completed

View in Issue Navigator

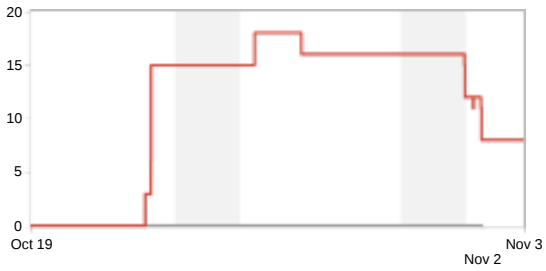
Key	Summary	Issue Type	Priority	Status	Story Points (3)
UXWEB-1 *	Basic project management(CRUD)	Story	Major	IN PROGRESS	3
UXWEB-4 *	UXWEB-2: LDAP login	Story	Major	IN PROGRESS	-
UXWEB-12 *	Set up server	Task	Major	IN PROGRESS	-

sUXess

Sprint Report

Sprint 2

Closed Sprint, ended by Roman Roba 19/Oct/15 1:00 PM - 03/Nov/15 7:12 PM Linked pages View Sprint 2 in Issue Navigator



Status Report

* Issue added to sprint after start time

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (3)
UXWEB-1 *	Basic project management(CRUD)	Story	Major	CLOSED	3
UXWEB-12 *	Set up server	Task	Major	CLOSED	-
UXWEB-39 *	Remove libraries from repository	Task	Major	CLOSED	-
UXWEB-41 *	GRUNT automation and SUBURI modification	Story	Major	CLOSED	-
UXWEB-44 *	JS comments methodics	Story	Major	CLOSED	-
UXWEB-45 *	RoR comments methodics + docs	Story	Major	CLOSED	-
UXWEB-47 *	RoR test methodics	Story	Major	CLOSED	-
UXWEB-51 *	Register for TP CUP	Story	Major	CLOSED	-
UXWEB-56 *	RoR methodics	Story	Major	CLOSED	-
UXWEB-60 *	Create a records for meeting 5 and 6	Task	Major	CLOSED	-

Issues Not Completed

View in Issue Navigator

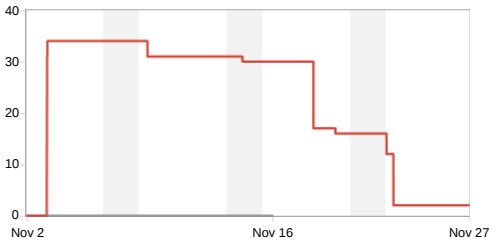
Key	Summary	Issue Type	Priority	Status	Story Points (1)
UXWEB-4 *	UXWEB-2: LDAP login	Story	Major	IN PROGRESS	-
UXWEB-46 *	JS tests methodics	Story	Major	OPEN	-
UXWEB-48 *	Deploy methodics	Story	Major	OPEN	-
UXWEB-49 *	Code review methodics	Story	Major	IN PROGRESS	-
UXWEB-50 *	Angular documentation methodics	Story	Major	OPEN	-
UXWEB-59 *	Jira methodics	Story	Major	IN PROGRESS	1

sUXess

Sprint Report

Sprint 3

Closed Sprint, ended by Roman Roba 02/Nov/15 3:30 PM - 27/Nov/15 6:17 PM Linked pages View Sprint 3 in Issue Navigator



Status Report

* Issue added to sprint after start time

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (29)
UXWEB-28 *	Image upload	Story	Major	CLOSED	3
UXWEB-46 *	JS tests methodics	Story	Major	CLOSED	1
UXWEB-48 *	Deploy methodics	Story	Major	CLOSED	1
UXWEB-49 *	Code review methodics	Story	Major	CLOSED	1
UXWEB-50 *	Angular documentation methodics	Story	Major	CLOSED	1
UXWEB-59 *	Jira methodics	Story	Major	CLOSED	1
UXWEB-61 *	Management Documentation	Story	Major	CLOSED	5
UXWEB-62 *	Work documentation	Story	Major	CLOSED	8
UXWEB-64 *	Dashboard template	Story	Major	CLOSED	2
UXWEB-65 *	Low/high fidelity web app template	Story	Major	CLOSED	5
UXWEB-66 *	Flash messages	Story	Major	CLOSED	1
UXWEB-69 *	Create a records for meeting 7, 8 and 9	Task	Major	CLOSED	-

Issues Not Completed

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (2)
UXWEB-63 *	Login template	Story	Major	IN PROGRESS	2

Issues Removed From Sprint

View in Issue Navigator

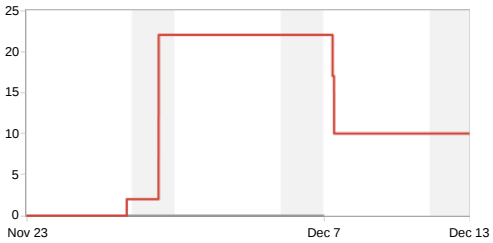
Key	Summary	Issue Type	Priority	Status	Story Points (3)
UXWEB-4 *	UXWEB-2: LDAP login	Story	Major	IN PROGRESS	3

sUXess

Sprint Report

Sprint 4

Closed Sprint, ended by Roman Roba 23/Nov/15 1:00 AM - 13/Dec/15 8:50 PM Linked pages View Sprint 4 in Issue Navigator



Status Report

* Issue added to sprint after start time

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (2)
UXWEB-63 *	Login template	Story	Major	CLOSED	2
UXWEB-70 *	Website template	Story	Major	CLOSED	-
UXWEB-72 *	Prototyp creation(CR) and Mockup creation(CRU)	Story	Major	CLOSED	-

Issues Not Completed

View in Issue Navigator

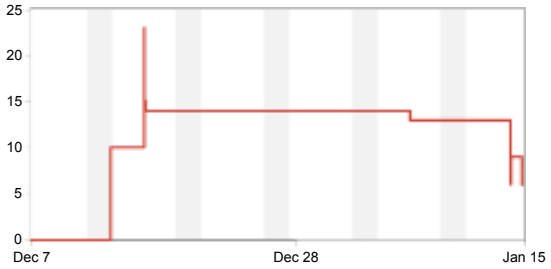
Key	Summary	Issue Type	Priority	Status	Story Points (5)
UXWEB-71 *	Modification and completition of documentation	Story	Major	OPEN	5
UXWEB-73 *	Mockup editor	Story	Major	IN PROGRESS	-

sUXess

Sprint Report

Sprint 5

Closed Sprint, ended by Roman Roba 07/Dec/15 1:00 PM - 15/Jan/16 2:57 PM Linked pages View Sprint 5 in Issue Navigator



Status Report

* Issue added to sprint after start time

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (11)
UXWEB-34 *	Basic test CRUD	Story	Major	CLOSED	-
UXWEB-71 *	Modification and completition of documentation	Story	Major	CLOSED	5
UXWEB-73 *	Mockup editor	Story	Major	CLOSED	5
UXWEB-77 *	Breadcrumb refactor	Story	Major	CLOSED	-
UXWEB-78 *	Mockup editor Tests	Story	Major	CLOSED	-
UXWEB-79 *	Mockup editor - Automatic size of image	Story	Major	CLOSED	1
UXWEB-80 *	Mockup editor - Mockup paging	Story	Major	CLOSED	-
UXWEB-81 *	Mockup editor - element update and delete	Story	Major	CLOSED	-
UXWEB-83 *	Styling - Dashboard	Story	Major	CLOSED	-
UXWEB-86 *	MIS/MSI presentation	Story	Major	CLOSED	-

Issues Not Completed

View in Issue Navigator

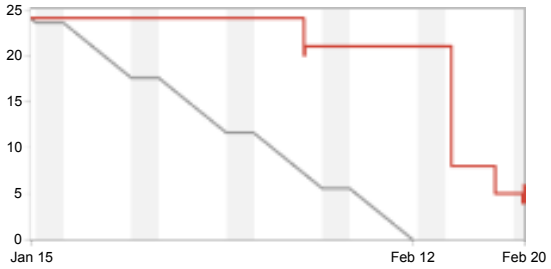
Key	Summary	Issue Type	Priority	Status	Story Points (3)
UXWEB-29 *	New test	Story	Major	IN PROGRESS	-
UXWEB-33 *	Tests list	Story	Major	IN PROGRESS	-
UXWEB-82 *	Styling - Login Page	Story	Major	IN PROGRESS	-
UXWEB-84 *	Styling - Prototypes	Story	Major	OPEN	-
UXWEB-85 *	Styling - Editor	Story	Major	IN PROGRESS	-
UXWEB-87 *	Task + Test	Story	Major	IN PROGRESS	3

sUXess

Sprint Report

Sprint 6

Closed Sprint, ended by Roman Roba 15/Jan/16 3:20 PM - 20/Feb/16 6:33 PM Linked pages View Sprint 6 in Issue Navigator



Status Report

* Issue added to sprint after start time

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (19)
UXWEB-29	New test	Story	Major	CLOSED	-
UXWEB-32	Test start	Story	Major	CLOSED	2
UXWEB-33	Tests list	Story	Major	CLOSED	-
UXWEB-35	Test display	Story	Major	CLOSED	5
UXWEB-74	Ability -> backend refactoring	Story	Critical	CLOSED	8
UXWEB-84	Styling - Prototypes	Story	Major	CLOSED	1
UXWEB-87	Task + Test	Story	Major	CLOSED	3

Issues Not Completed

View in Issue Navigator

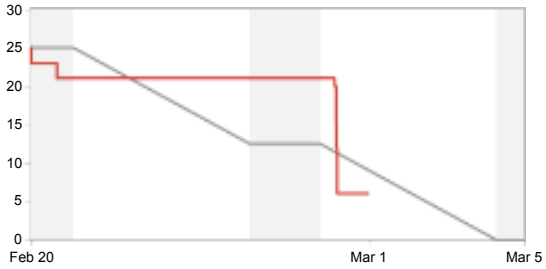
Key	Summary	Issue Type	Priority	Status	Story Points (5)
UXWEB-82	Styling - Login Page	Story	Major	IN PROGRESS	1
UXWEB-85	Styling - Editor	Story	Major	IN PROGRESS	1
UXWEB-88	Create a test URL	Story	Major	IN PROGRESS	3
UXWEB-89 *	Styling - Task	Story	Major	IN PROGRESS	-

sUXess

Sprint Report

Sprint 7

Closed Sprint, ended by Roman Roba 20/Feb/16 6:57 PM - 01/Mar/16 9:19 AM Linked pages View Sprint 7 in Issue Navigator



Status Report

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (19)
UXWEB-82	Styling - Login Page	Story	Major	CLOSED	1
UXWEB-85	Styling - Editor	Story	Major	CLOSED	1
UXWEB-88	Create a test URL	Story	Major	CLOSED	3
UXWEB-89	Styling - Task	Story	Major	CLOSED	1
UXWEB-90	Capturing user actions	Story	Major	CLOSED	3
UXWEB-91	Sending + processing of captured user data	Story	Major	CLOSED	5
UXWEB-92	Defining target element	Story	Major	CLOSED	3
UXWEB-95	IIT.SRC	Story	Major	CLOSED	2

Issues Not Completed

View in Issue Navigator

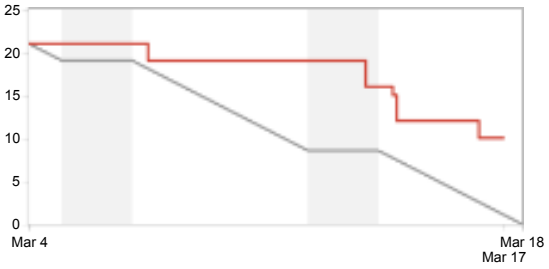
Key	Summary	Issue Type	Priority	Status	Story Points (6)
UXWEB-93	Link tool - editor	Story	Major	IN PROGRESS	3
UXWEB-94	Preview/Show test	Story	Major	IN PROGRESS	3

sUXess

Sprint Report

Sprint 8

Closed Sprint, ended by Roman Roba 04/Mar/16 1:44 AM - 17/Mar/16 1:47 PM Linked pages View Sprint 8 in Issue Navigator



Status Report

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (11)
UXWEB-93	Link tool - editor	Story	Major	RESOLVED	3
UXWEB-94	Preview/Show test	Story	Major	CLOSED	3
UXWEB-96	Adding of "empty element"	Story	Major	RESOLVED	1
UXWEB-97	Launching of the test	Story	Major	RESOLVED	2
UXWEB-99	Eventer	Story	Major	RESOLVED	2

Issues Not Completed

View in Issue Navigator

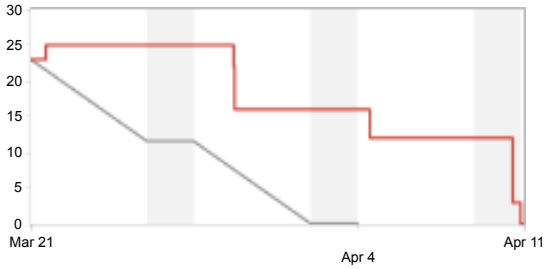
Key	Summary	Issue Type	Priority	Status	Story Points (10)
UXWEB-98	Preview and test run of the test.	Story	Major	IN PROGRESS	1
UXWEB-100	Update and delete of mockups.	Story	Major	IN PROGRESS	2
UXWEB-101	Evaluate the completion of the task	Story	Major	IN PROGRESS	7

sUXess

Sprint Report

Sprint 9

Closed Sprint, ended by Roman Roba 21/Mar/16 12:49 AM - 11/Apr/16 3:00 AM Linked pages View Sprint 9 in Issue Navigator



Status Report

Completed Issues

View in Issue Navigator

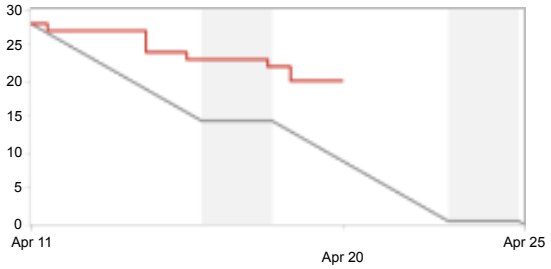
Key	Summary	Issue Type	Priority	Status	Story Points (23)
UXWEB-98	Preview and test run of the test.	Story	Major	CLOSED	1
UXWEB-100	Update and delete of mockups.	Story	Major	CLOSED	2
UXWEB-101	Evaluate the completion of the task	Story	Major	CLOSED	7
UXWEB-105	Heat Maps (clicks)	Story	Major	CLOSED	5
UXWEB-106	Showing results	Story	Major	RESOLVED	3
UXWEB-107	Editor - fixed resolution	Story	Major	CLOSED	2
UXWEB-108	Test Run	Story	Major	RESOLVED	2
UXWEB-109	Test End	Story	Major	RESOLVED	1
UXWEB-110	Project Overview with Tests Overview	Story	Major	RESOLVED	-

sUXess

Sprint Report

Sprint 10

Closed Sprint, ended by Roman Roba 11/Apr/16 3:34 AM - 20/Apr/16 12:20 AM Linked pages View Sprint 10 in Issue Navigator



Status Report

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (8)
UXWEB-114	Test - Performing	Story	Major	CLOSED	2
UXWEB-117	Logging of tester browser.	Story	Major	RESOLVED	1
UXWEB-118	Dashboard - Number of test participants	Story	Major	RESOLVED	1
UXWEB-119	Scipts(Tests, Server run, Sidekiq, Redis)	Story	Major	RESOLVED	1
UXWEB-124	Surveys	Story	Major	RESOLVED	3

Issues Not Completed

View in Issue Navigator

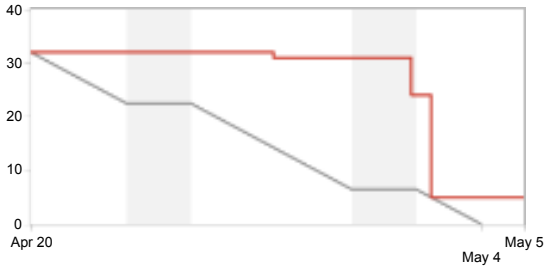
Key	Summary	Issue Type	Priority	Status	Story Points (20)
UXWEB-113	Results improvements	Story	Major	IN PROGRESS	5
UXWEB-115	Editor styling	Story	Major	IN PROGRESS	5
UXWEB-116	Test Edit	Story	Major	IN PROGRESS	1
UXWEB-120	404 screen - Styling	Story	Major	OPEN	1
UXWEB-121	Eye tracker	Story	Major	IN PROGRESS	5
UXWEB-122	Metrics	Story	Major	OPEN	2
UXWEB-123	Mouse movements	Story	Major	OPEN	1

sUXess

Sprint Report

Sprint 11

Closed Sprint, ended by Roman Roba 20/Apr/16 12:34 AM - 05/May/16 8:26 AM Linked pages View Sprint 11 in Issue Navigator



Status Report

Completed Issues

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (27)
UXWEB-113	Results improvements	Story	Major	RESOLVED	5
UXWEB-115	Editor styling	Story	Major	CLOSED	5
UXWEB-116	Test Edit	Story	Major	CLOSED	1
UXWEB-121	Eye tracker	Story	Major	CLOSED	5
UXWEB-122	Metrics	Story	Major	CLOSED	2
UXWEB-123	Mouse movements	Story	Major	CLOSED	1
UXWEB-127	ITT.SRC Poster	Story	Major	CLOSED	3
UXWEB-128	Login redirection bug	Story	Major	RESOLVED	1
UXWEB-129	Filter (Query Object)	Story	Major	CLOSED	2
UXWEB-130	ITT.SRC - Preparations	Story	Major	RESOLVED	2

Issues Not Completed

View in Issue Navigator

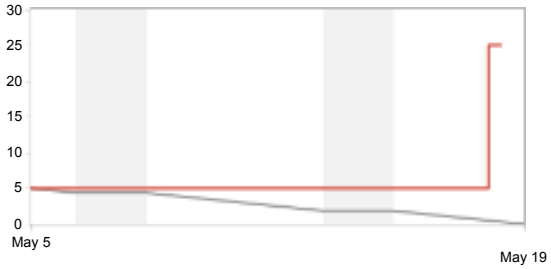
Key	Summary	Issue Type	Priority	Status	Story Points (5)
UXWEB-131	Web screenshots	Story	Major	OPEN	5

sUXess

Sprint Report

Sprint 12

Active Sprint 05/May/16 4:59 PM - 19/May/16 4:59 PM Linked pages View Sprint 12 in Issue Navigator



Status Report

* Issue added to sprint after start time

Issues Not Completed

View in Issue Navigator

Key	Summary	Issue Type	Priority	Status	Story Points (10)
UXWEB-131	Web screenshots	Story	Major	OPEN	5
UXWEB-133 *	Documentation of management	Story	Major	IN PROGRESS	-
UXWEB-134 *	Documentation of the Project	Story	Major	OPEN	-
UXWEB-135 *	UX testing - feedback	Story	Major	OPEN	5
UXWEB-136 *	Finish missing tests	Story	Major	OPEN	-
UXWEB-137 *	Merge + Deploy	Story	Major	OPEN	-

D Inštalačná príručka

V tejto príručke sú vymenované všetky závislosti potrebné pre inštaláciu a spustenie nášho projektu, ako aj spísaný postup inštalácie.

D.1 Inštalácia a spustenie projektu

Pre lokálne spustenie celého projektu, backend aj frontend, ktoré existujú v tom istom repozitári, je potrebné mať nainštalované nasledujúce systémové služby:

- Ruby (minimálne verzia 2.0)[4],
- PostgreSQL[5],
- Redis[6],
- Node.js (npm)[7].

Postup ich inštalácie si je potrebné pozrieť na ich domovských stránkach, prípadne, pri použití niektorej z GNU/Linux distribúcií, sa riadiť odporúčaním danej distribúcie.

Zvyšná časť inštalácie sa odohráva v koreňovom priečinku repozitára projektu. Je potrebné nainštalovať bundler[8] pre správu závislostí Ruby projektov:

```
$ gem install bundler
```

Na inštaláciu Rails-ov a zvyšných Ruby GEM-ov a spustenie migrácie databázy treba spustiť:

```
$ bundle install
$ bundle exec rake db:migrate
```

Ďalej je potrebné ešte nainštalovať všetky závislosti frontendu. Na to je potrebný bower. Inštalácia prebieha v priečinku `public`, kde sa frontend nachádza:

```
$ cd public
$ npm install bower
$ bower install
```

Na následné spustenie celej aplikácie, je potrebné mať spustený PostgreSQL a Redis. Je potrebné spustiť Sidekiq, ktorý vyhodnocuje výsledky vykonaných testov v našom projekte, a spustiť Rails sever:

```
$ bundle exec sidekiq
$ bundle exec rails s
```

D.2 Príprava vývojárskeho prostredia

Dodatočne, pre vývojárske potreby, je možné nainštalovať, ktoré sú použité pri vývoji frontendu. V `public` zložke:

[4] `https://www.ruby-lang.org/`
[5] `http://www.postgresql.org/`
[6] `http://redis.io/`
[7] `https://nodejs.org/`
[8] `http://bundler.io/`

```
$ npm install
$ grunt package:development
$ npm install
```

Npm inštaluje na základe súboru `package.json`. Základný súbor obsahuje závislosti na Grunt[9] a niektoré plug-in-y doň. `grunt package:development` príkaz zamení `package.json` súbor za viac rozsiahlejší, kde sú už všetky závislosti potrebné pre kompiláciu Sass a spustenie frontend testov.

D.3 Nasadenie do produkčného prostredia

Pri nasadzovaní do produkčného prostredia sa používa Capistrano[10]. Jeho konfigurácia sa nachádza v týchto dvoch súboroch: `config/deploy.rb` a `config/deploy/production.rb`. Capistrano vykonáva deploy cez SSH. Je nakonfigurané tak, že očakáva, aby na serveri bolo použité RVM[11] na správu Ruby verzií, bežiaci aplikačný server Passenger[12] a bežiaci Sidekiq.

Konfiguráciu Capistran-a bude potrebné upraviť pre potreby iného produkčného servera, pretože využíva premenné prostredia, `SERVER_IP` a `WERCKER_GIT_COMMIT`, ktoré sú dostupné v prostredí kontinuálnej integrácie, ktorá nasadenie spúšťa. Takisto je v nastavenia špecifikovaný RVM gemset, ktorý sa očakáva na serveri, a URL repozitára, odkiaľ sa zdrojové kódy stiahnu.

Pre spustenie nasadenia sa spúšta príkaz:

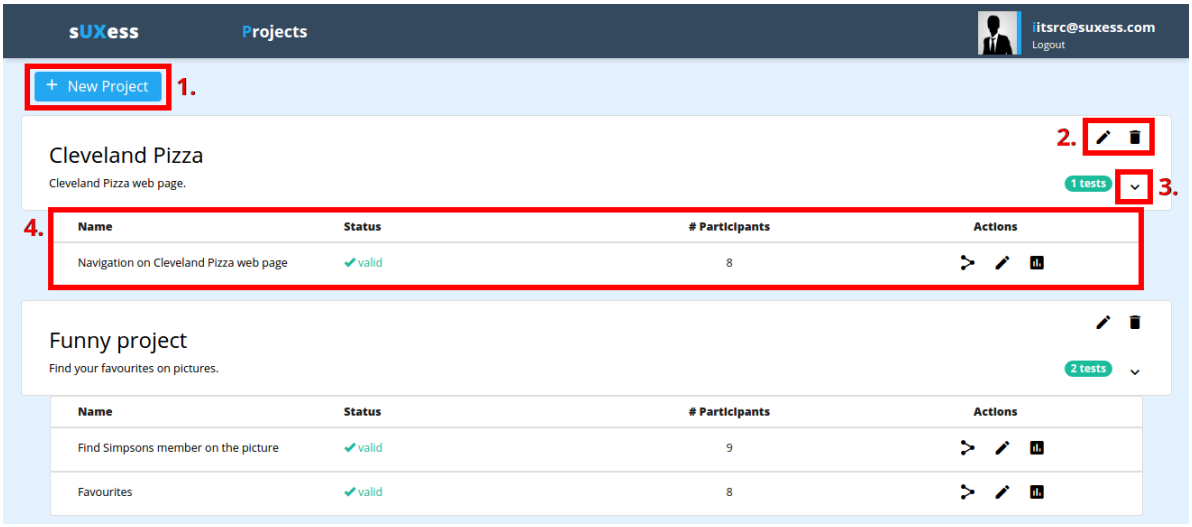
```
$ bundle exec cap production deploy
```

[9] `http://gruntjs.com/`
[10] `http://capistranorb.com/`
[11] `https://rvm.io/`
[12] `https://www.phusionpassenger.com/`

E Používateľská príručka

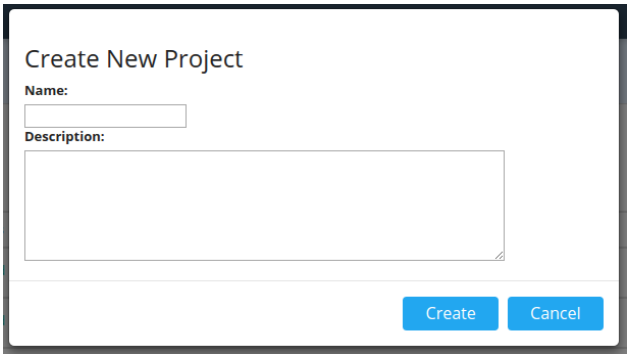
E.1 Správa projektov

Keď sa používateľ prihlási, je presmerovaný na stránku so zoznamom všetkých jeho projektov. Na obrázku 25 môžeme vidieť príklad takejto stránky. Každý projekt má svoj názov a popis, ktoré sú v tomto výpise zobrazené. Používateľ môže cez tlačidlá, označené bodom 2, upraviť alebo odstrániť daný projekt. Používateľ si môže zobraziť stručný výpis testov, ktoré vrámci projektu existujú (bod 3 a 4). Má tak prehľad o tom, v akom stave sú jednotlivé testy a koľko ľudí sa ich už zúčastnilo.



Obr. 25: Obrazovka zobrazujúca prehľad používateľových projektov.

Pre pridanie nového projektu je potrebné kliknúť na tlačidlo “New Project” (bod 1). Následne sa zobrazí formulár ako na obrázku 26. Je potrebné tam vyplniť názov projektu a jeho popis.

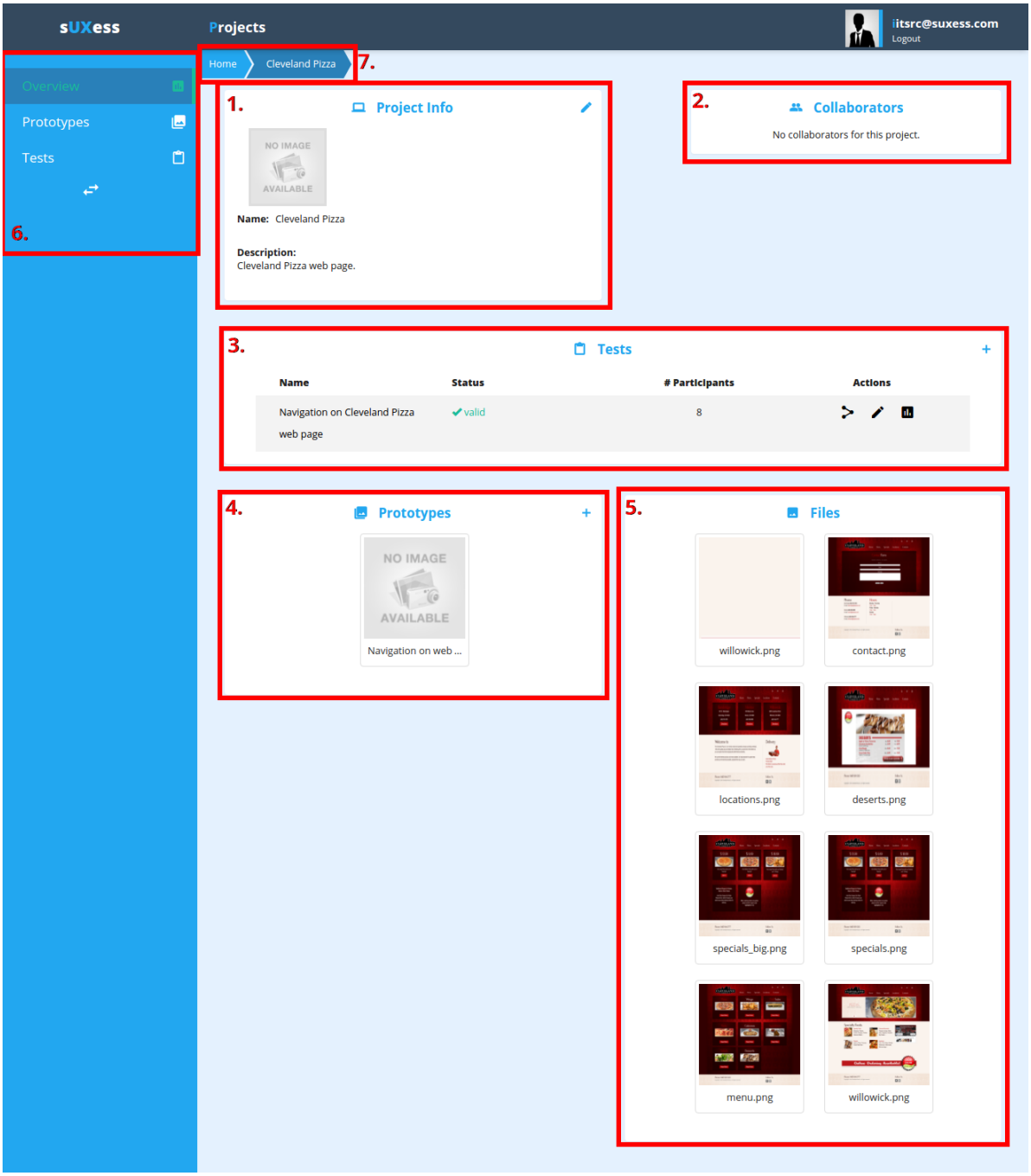


Obr. 26: Formulár pre vytvorenie nového projektu.

Po kliknutí na projekt je zobrazený prehľad projektu ako na obrázku 27. Prehľad sa skladá z viacerých častí (body súhlasia s číslami v obrázku):

1. *informácie o projekte*: je tu zobrazený názov projektu a jeho opis. Po kliku na ikonku editovania v pravo hore, je možné priamo na mieste upraviť detaily projektu;

2. *kolaborátori*: je tu zobrazený zoznam kolaborátorov, ktorí sa na projekte podiaľajú;



Obr. 27: Obrazovka stránky prehľadu projektu.

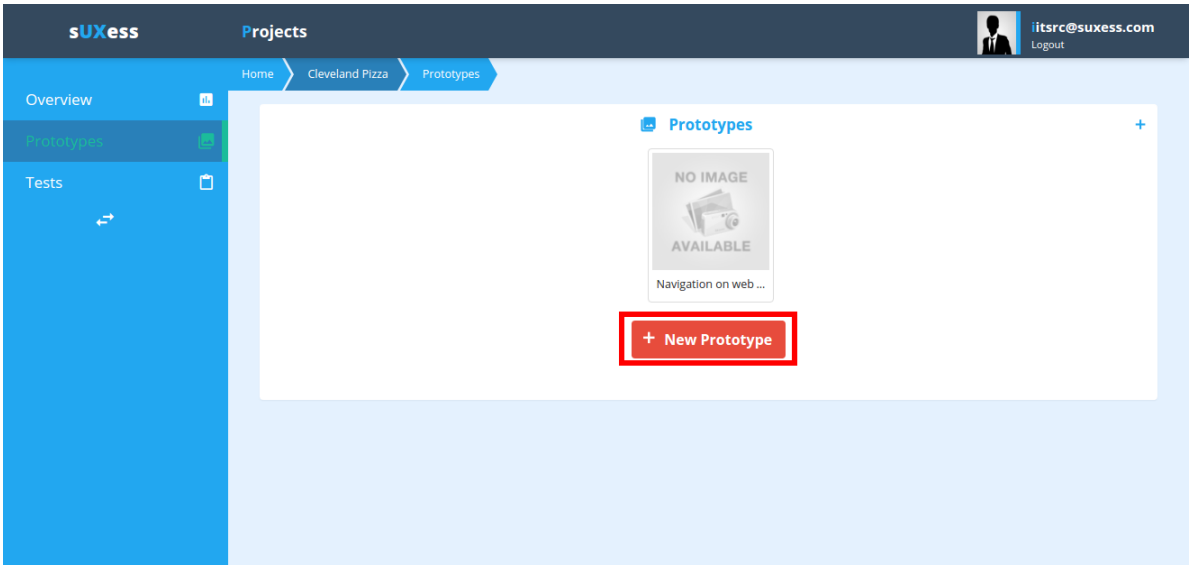
3. *testy*: podobne, ako na obrazovke projektov (obrázok 25), aj tu je prehľadný zoznam testov, ktoré sa v danom projekte nachádzajú a základné informácie o nich. Sú tu aj tlačidlá (sprava doľava) pre zdieľanie linku testu, úpravu testu a tlačidlo na presun na stránku s výsledkami daného testu;
4. *prototypy*: stručný náhľad na prototypy, ktoré vrámci projektu existujú;
5. *súbory*: stručný prehľad súborov, ktoré sú v projekte použité;
6. *bočný panel*: slúži na pohybovanie sa vrámci projektu. Spodná ikonka ho umožňuje zmenšiť. Umožňuje

nám ísť na obrazovku prehľadu, na obrazovku na správu prototypov (podrobnejšie v sekcii E.2) a na prehľad testov (viac v sekcii E.3);

7. *aktuálna cesta*: zobrazuje, kde v projekte sa práve nachádzame.

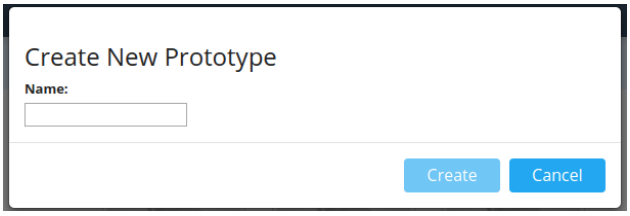
E.2 Správa prototypov

Na obrazovke pre správu prototypov (obrázok 28) je možné vidieť prehľad existujúcich prototypov, ktoré sa v projekte nachádzajú. Každý prototyp má svoje meno.



Obr. 28: Obrazovka súhrnu prototypov, ktoré boli v projekte vytvorené.

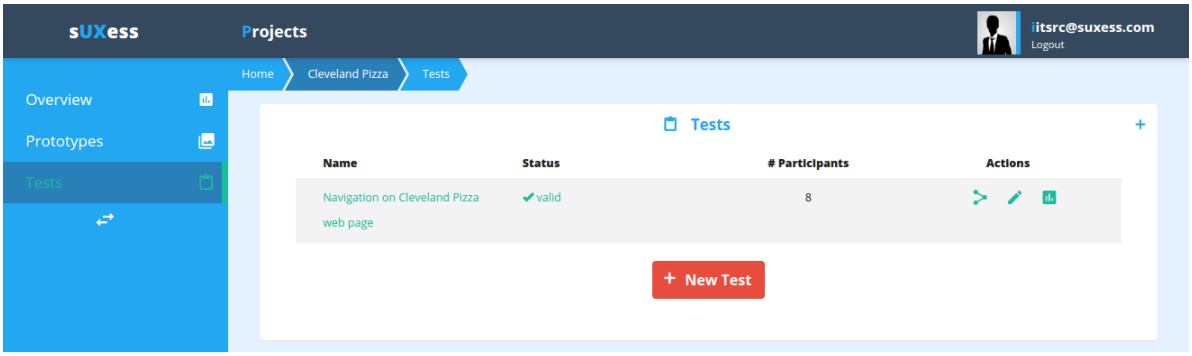
Po kliknutí na tlačidlo pre pridanie prototypu sa zobrazí formulár ako na obrázku 29, v ktorom je potrebné vyplniť názov a môžeme vytvoriť prototyp. Po dvojitom kliknutí na prototyp sa nám spustí editor, ktorý je podrobnejšie opísaný v prílohe F.



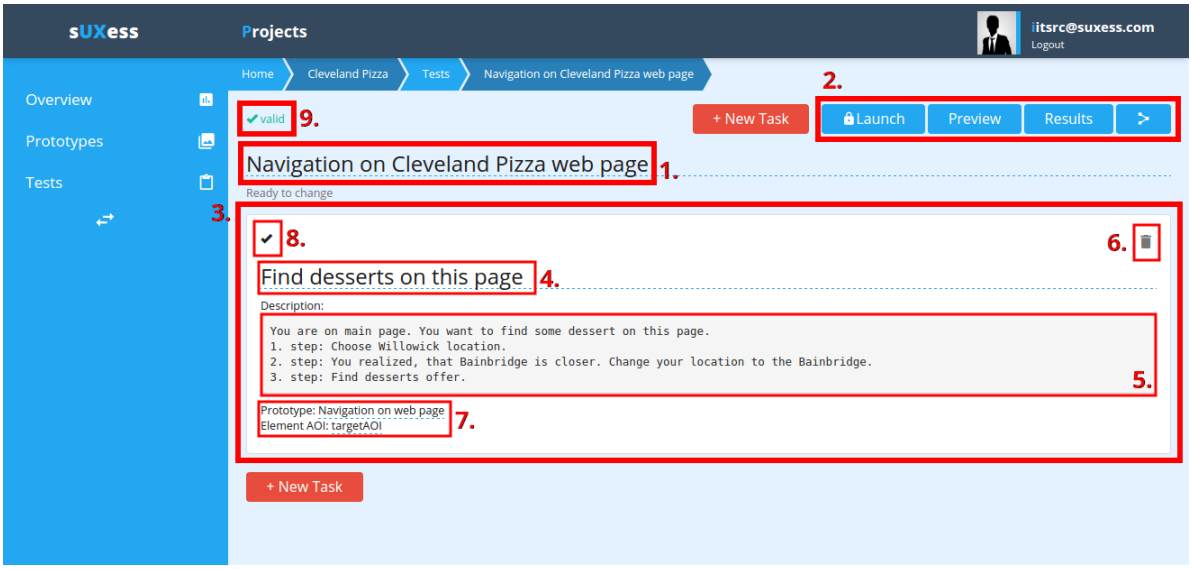
Obr. 29: Formulár pre pridanie nového prototypu.

E.3 Správa testov

Na obrázku 30 je možné vidieť obrazovku prehľadu testov. Je to podobný zoznam testov, ako sme videli na domovskej obrazovke a aj v prehľade projektu. Je tu možné kliknúť na pridanie nového testu alebo kliknúť na už existujúci test. V oboch prípadoch sa nám zobrazí obrazovka podobná ako na obrázku 31.

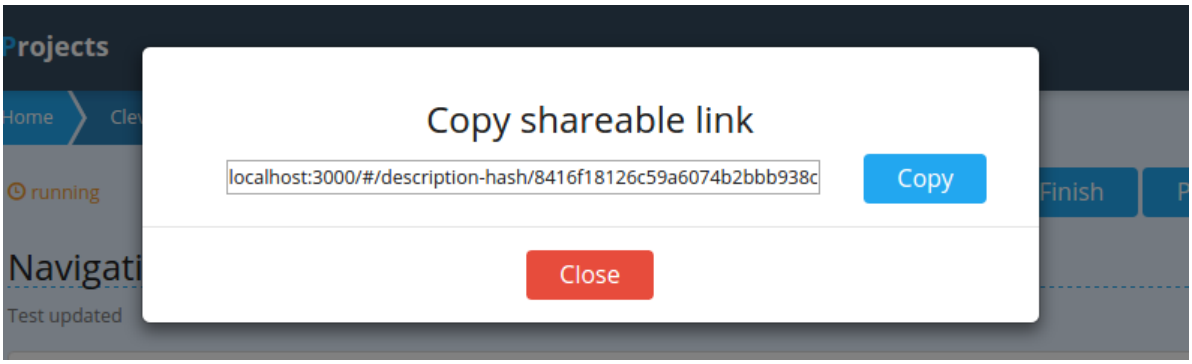


Obr. 30: Obrazovka prehľadu testov.



Obr. 31: Obrazovka detailu testu.

Na tejto obrazovke môžeme vidieť názov testu (bod 1), ktorý je možné po kliknutí naň zmeniť. Pri bode 2 je možné vidieť súbor tlačidiel. Prvé slúži na spustenie testu, druhé na zobrazenie náhľadu na to, ako bude test vyzerať, tretie na zobrazenie výsledkov k danému testu a posledné tlačidlo umožňuje zobraziť formulár, kde sa používateľovi vygeneruje link na zdieľanie testu s testermi (obrázok 32).



Obr. 32: Formulár pre zdieľanie linku na test.

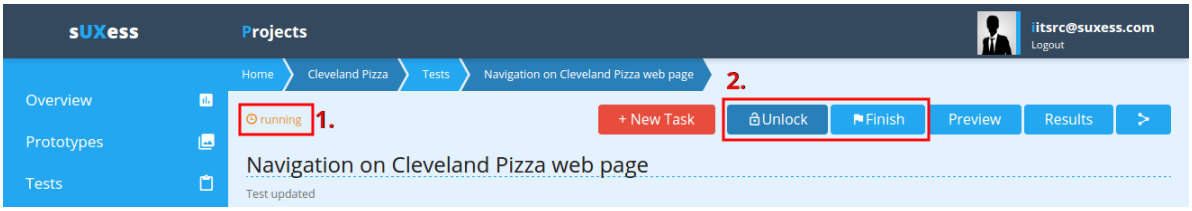
E.3.1 Tasky

Každý test sa skladá z určitého počtu taskov. Ako je možné vidieť na obrázku 31, všetky tasky sú vypísané na obrazovke pre detail testu (bod 3). Každý task má názov a opis (body 4 a 5) a taktiež prototyp, ktorý používa, a môže mať nastavený aj element, ktorý určuje cieľovú oblasť záujmu (bod 7). Všetky tieto položky je možné upravovať priamo na mieste po kliknutí na ne. Cez tlačidlo v pravom hornom rohu (bod 6) je možné task zmazať. Ak má task vyplnený názov, popis a pridelený prototyp, je tento task označený ako validný (bod 8).

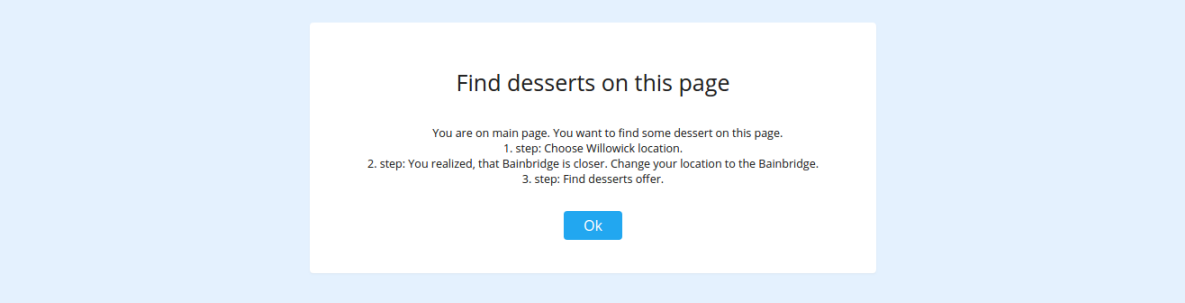
E.3.2 Priebeh testu

Ak sú všetky tasky testu validné, potom aj test je označený ako validný. Až vtedy je možné spustiť test a znemožniť jeho ďalšie jeho úpravy cez prvé tlačidlo v bode 2 na obrázku 31. Po spustení testu, ako je možné vidieť na obrázku 33, sa zmení jeho stav na bežiaci (bod 1). Používateľ má potom možnosť test odomknúť a znova ho upraviť alebo ho ukončiť (bod 2). Musí pritom brať na vedomie, že ak ho chce znova editovať, test už mohol byť testermi vykonaný a po zmene môžu byť výsledky testu znehodnotené. Ukončenie testu znamená, že nebude už možné ďalšie jeho testovanie testermi.

Ako už bolo spomenuté, vlastník testu umožní testovanie cez link. Po kliknutí na tento link sa spustí testovanie. Striedajú sa tam dve verzie obrazoviek. Pred každým taskom z testu je zobrazený jeho popis ako na obrázku 34. Tester má čas na prečítanie si inštrukcií a keď vie, čo má robiť, môže kliknúť na “OK” pre pokračovanie.

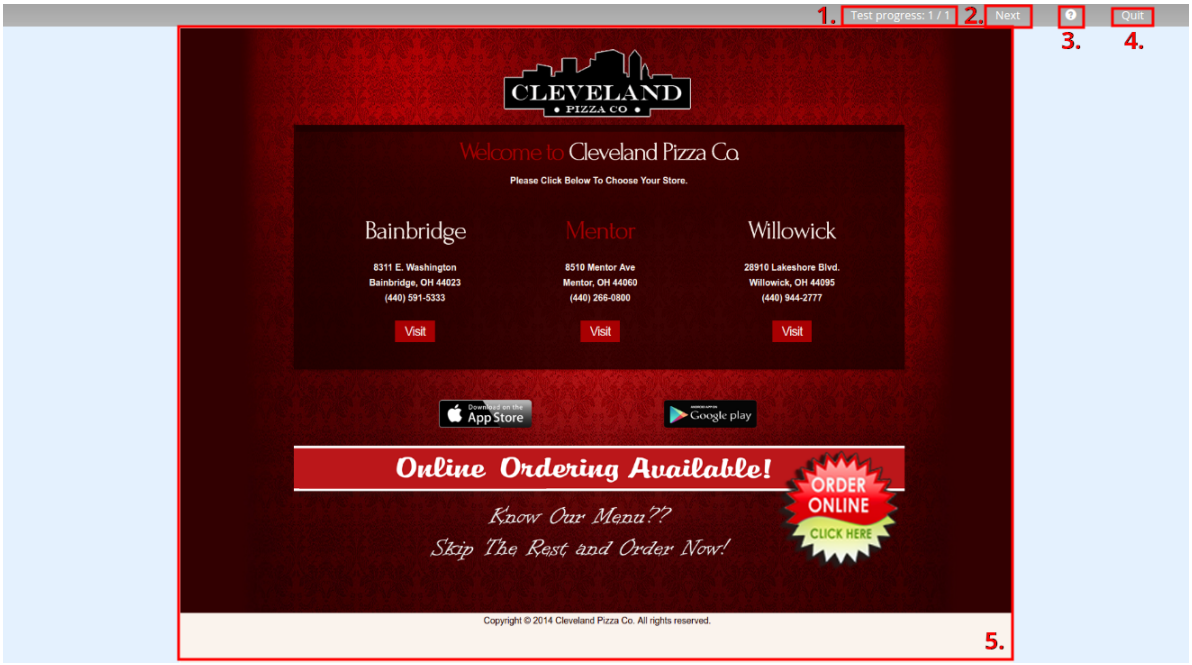


Obr. 33: Horná časť obrazovky po uzamknutí testu.



Obr. 34: Obrazovka opisu tasku pri vykonávaní testu.

Následne je mu už zobrazená obrazovka tasku, obrázok 35. Skladá sa z prototypu, ktorý bol danému tasku priradený, ktorý tvorí hlavnú časť tasku (bod 5). V hornom paneli, bod 1, je zobrazená informácia o tom, v ktorej časti testu sa tester nachádza. Tlačidlo "Next" (bod 2) umožňuje prejsť na nasledujúci task. Ak aktuálny task bol posledný, test sa ukončí, ak nasleduje ďalší task, je opäť zobrazená obrázka s opisom tasku. Ikonka v bode 3 slúži ako nápoveda pre testera v prípade, že si nie je istý tým, aká bola jeho úloha. Môže si tam opätovne zobraziť informáciu k tasku. Tlačidlo "Quit" (bod 4) test ukončí.



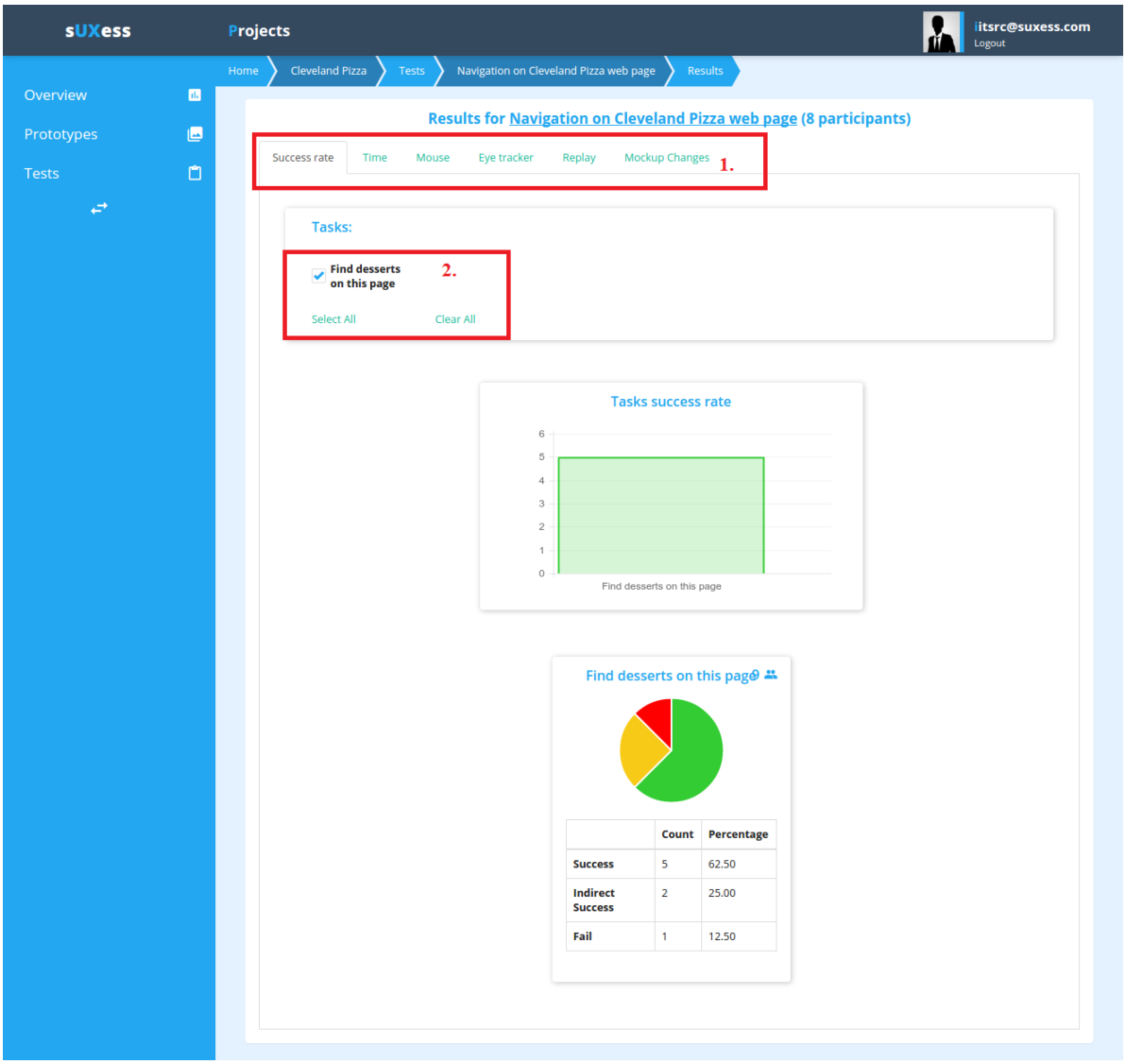
Obr. 35: Obrazovka tasku pri vykonávaní testu.

E.3.3 Náhľad testu

Používateľ si pred spustením testu môže vyskúšať ako bude vyzerať reálny test po spustení, čiže čo presne budú vidieť jeho testeri. Túto funkcionalitu možno vidieť na obrázku 31, na ktorom sa v bode 2 nachádza tlačidlo "Preview". Po stlačení tohto tlačidla sa zobrazí náhlad, ktorého vzhľad a funkcionalita je identická s opisom v kapitole E.3.2, kde sa striedajú obrazovky s popisom tasku 34 a obrazovky, na ktorých prebieha test 35.

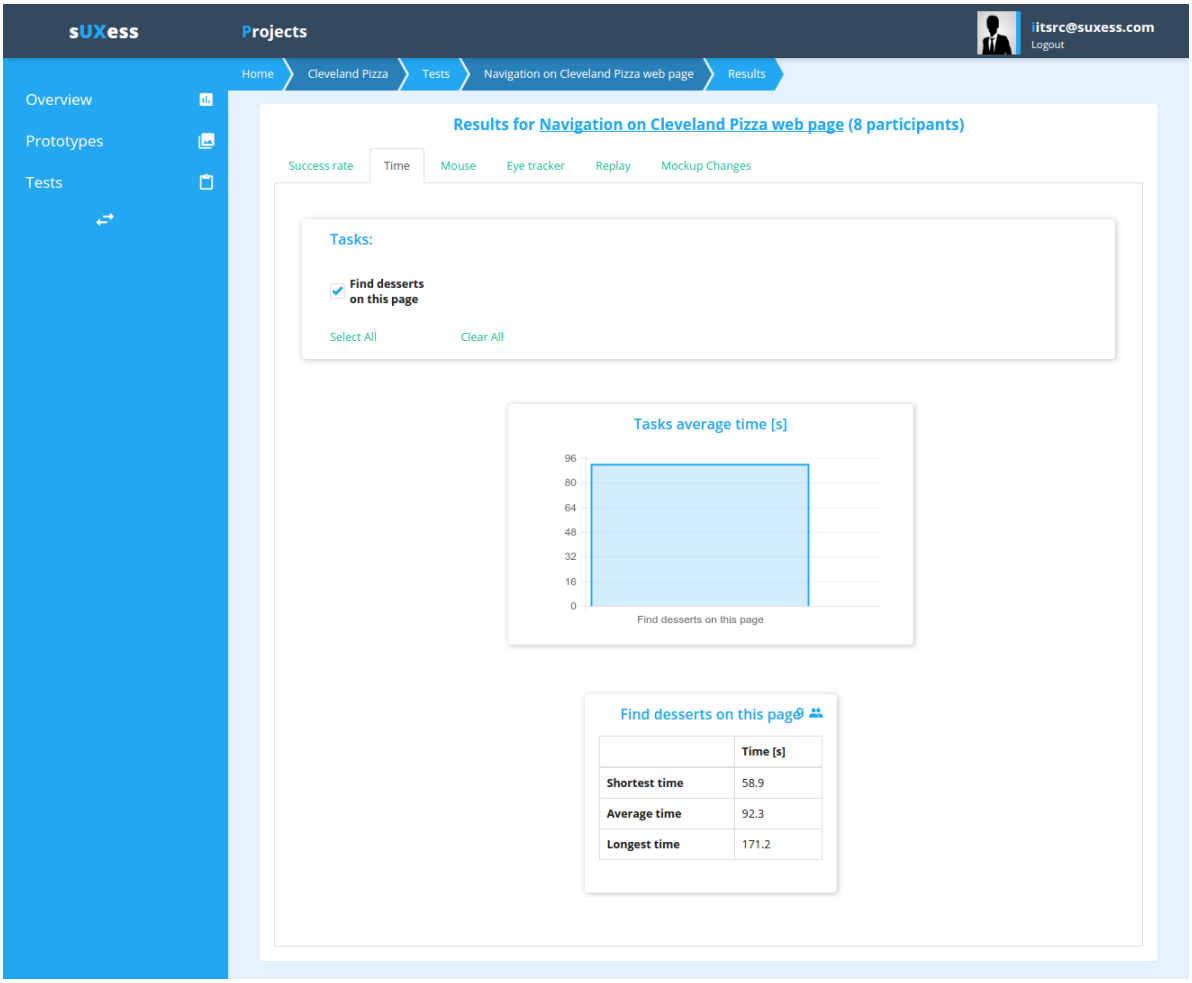
E.4 Výsledky testov

Po kliknutí na tlačidlo “Results”, ktoré možno vidieť na obrázku 31 v bode 2, sa používateľovi zobrazí obrazovka s výsledkami 36, ktoré sa týkajú daného testu. V bode 1 na tomto obrázku sa nachádzajú všetky typy výsledkov, ktoré sa zobrazia po kliknutí na danú kategóriu. V bode 2 možno vidieť kritéria, podľa ktorých možno filtrovať zobrazované výsledky, v tomto prípade zvolenie testov.



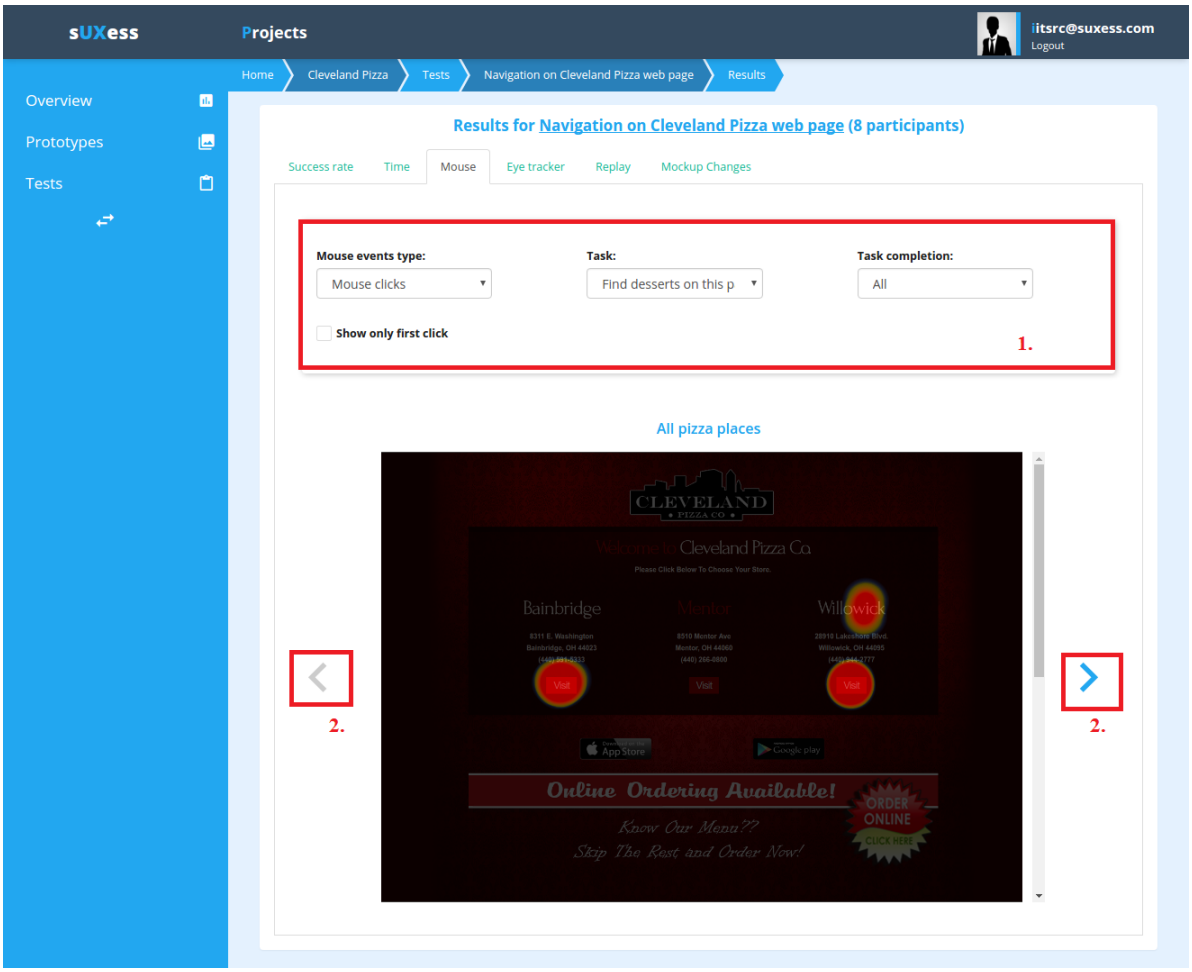
Obr. 36: Obrazovka pre úspešnosť vykonávania taskov.

Po kliknutí na tlačidlo “Time” sa zobrazia štatistiky týkajúce sa času vykonávania jednotlivých taskov 37.



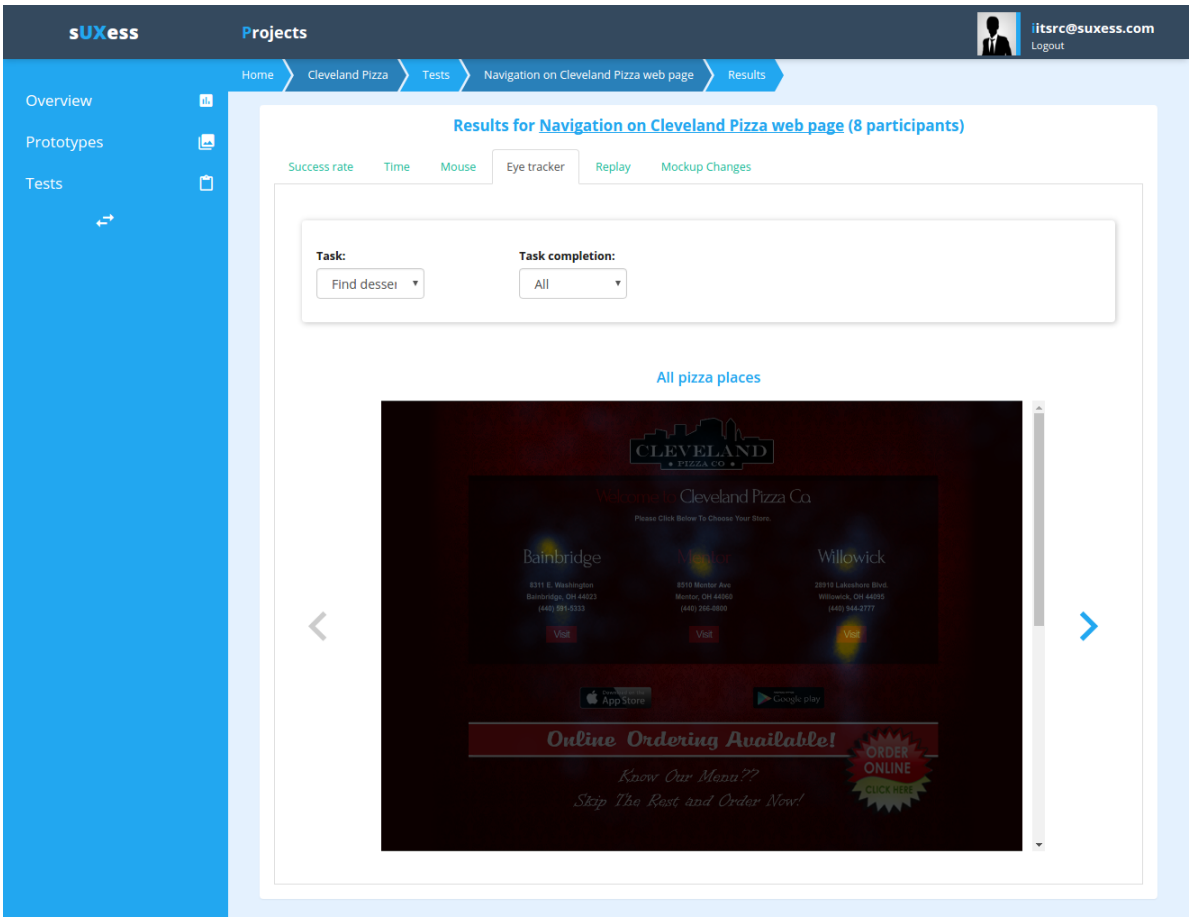
Obr. 37: Obrazovka so štatistikami časov pre tasky.

Na obrázku 38 vidno heatmapy týkajúce sa interakcie testerov prostredníctvom myši, kde v bode 1 možno filtrovať zobrazované výsledky podľa aktivity myši, ktorá môže byť buď kliky alebo pohyby, ďalej podľa jednotlivých taskov, podľa ich úspešnosti a pre kliky možno zvoliť aj zobrazenie iba prvých klikov na danom mockupe. V bode 2 sa nachádzajú šípky, pomocou ktorých možno prepínať medzi mockupmi (obrazovkami) daného tasku.



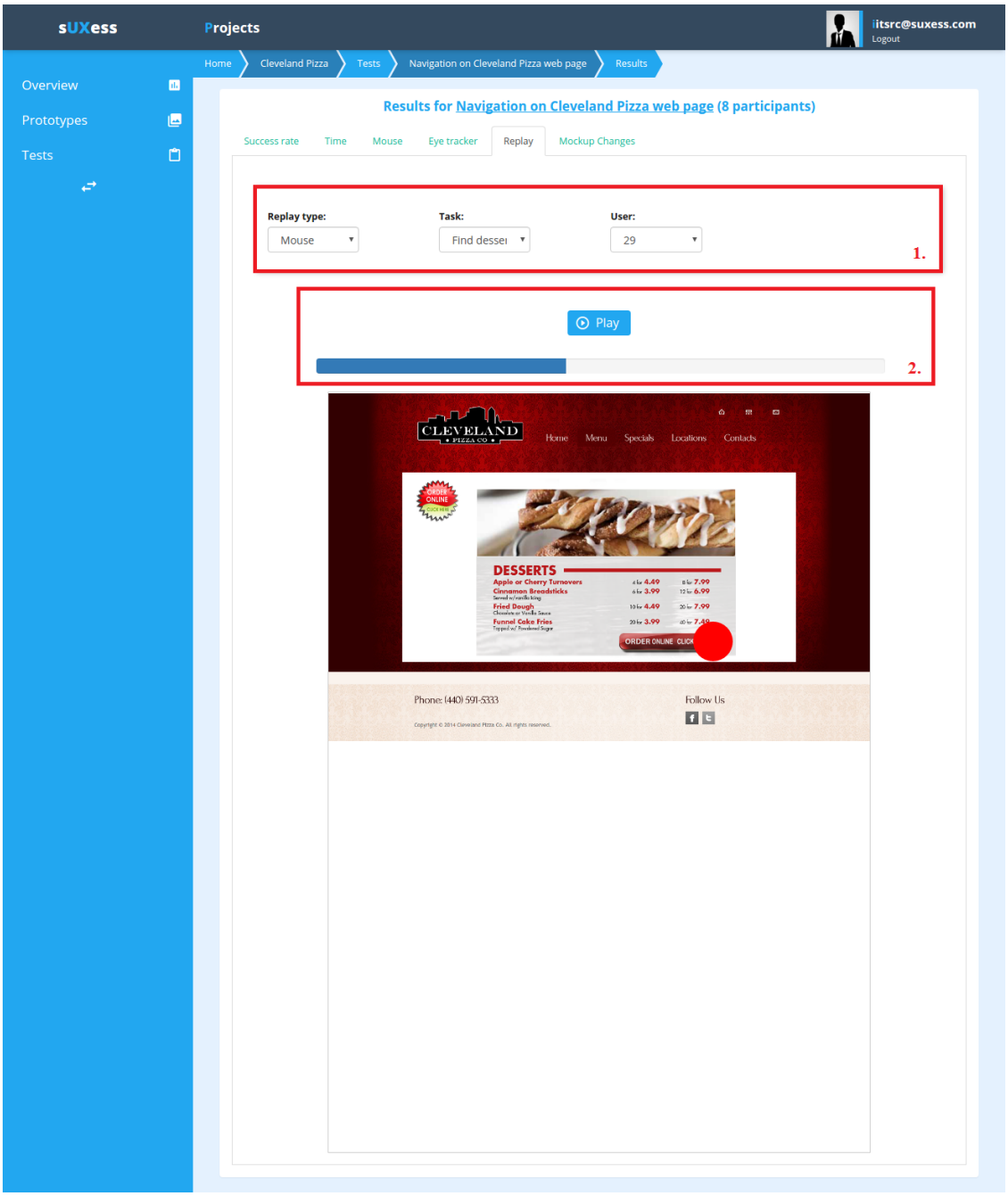
Obr. 38: Obrazovka so štatistikami pre interakciu s myšou.

V sekcií “Eye tracker” 39 sa nachádzajú heatmapy, z dát zozbieraných zo snímačov pohľadu, kde podobne ako na obrázku 38 možno filtrovať dáta a prepínať medzi mockupmi.



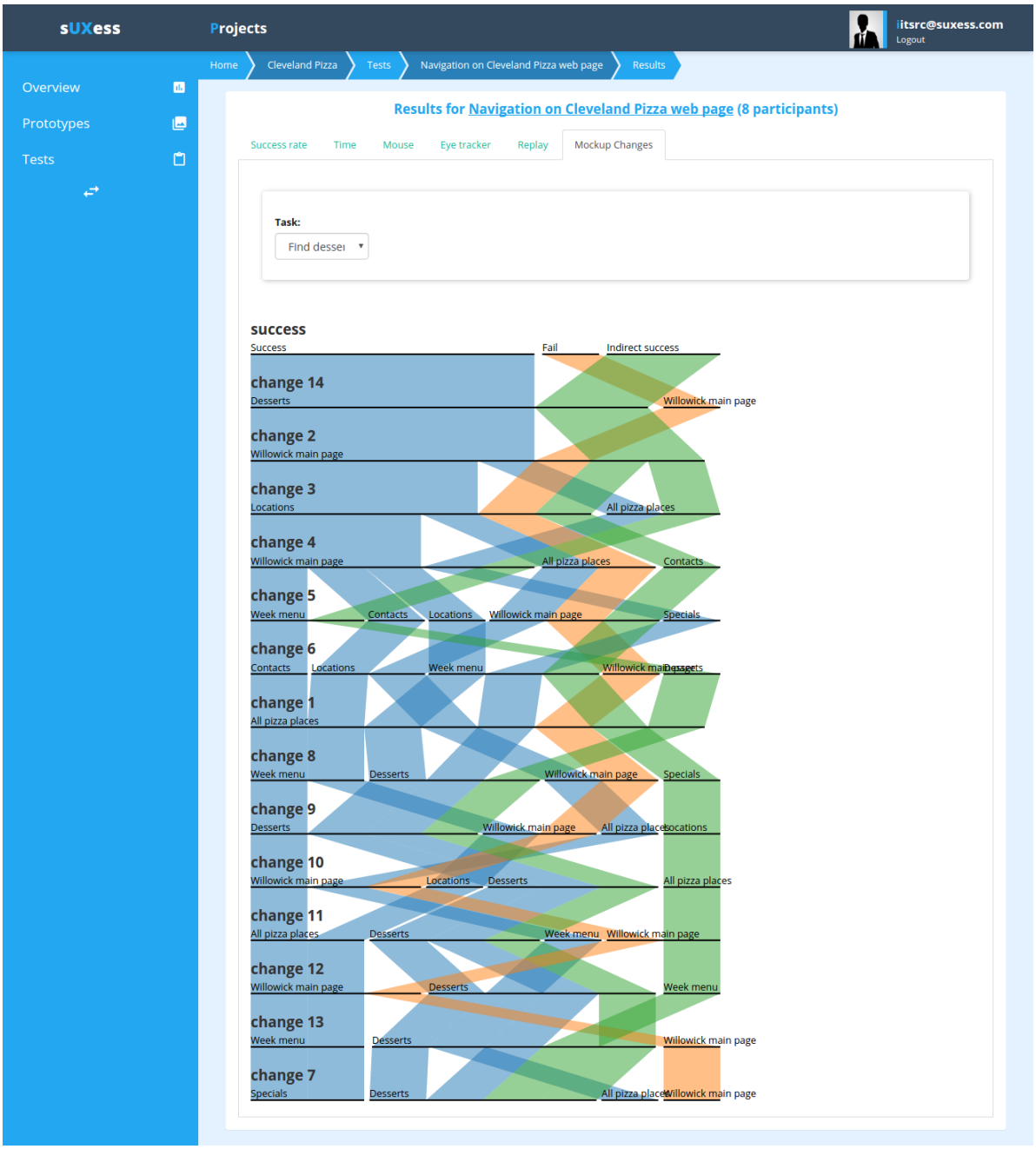
Obr. 39: Obrazovka so štatistikami zozbieranými prostredníctvom eyetrackera.

Po stlačení tlačidla “Replay” 40 sa zobrazí obrazovka, na ktorej si možno prehrať presnú aktivitu konkrétneho testera, čiže ako sa pohyboval a klikal s myšou a kam sa pozeral. V bode 1 si používateľ zvolí, či chce zobrazovať nahrávku pre dáta z myši alebo zo snímačov pohľadu, zvolí si ktorý task, a taktiež používateľa, ktorého nahrávku chce vidieť. V bode číslo 2 sa nachádza časová os pre zvolenú nahrávku a tlačidlo “Play”, ktoré spúšťa prehratie nahrávky.



Obr. 40: Obrazovka určená na rekonštrukciu aktivity jednotlivých používateľov.

V sekcií “Mockup changes” 41 sa nachádza graf, ktorý vyjadruje ako sa pohybovali používatelia medzi jednotlivými mockupmi tasku, čiže zobrazuje prechody medzi mockupmi.



Obr. 41: Obrazovka s prechodmi medzi mockupmi.

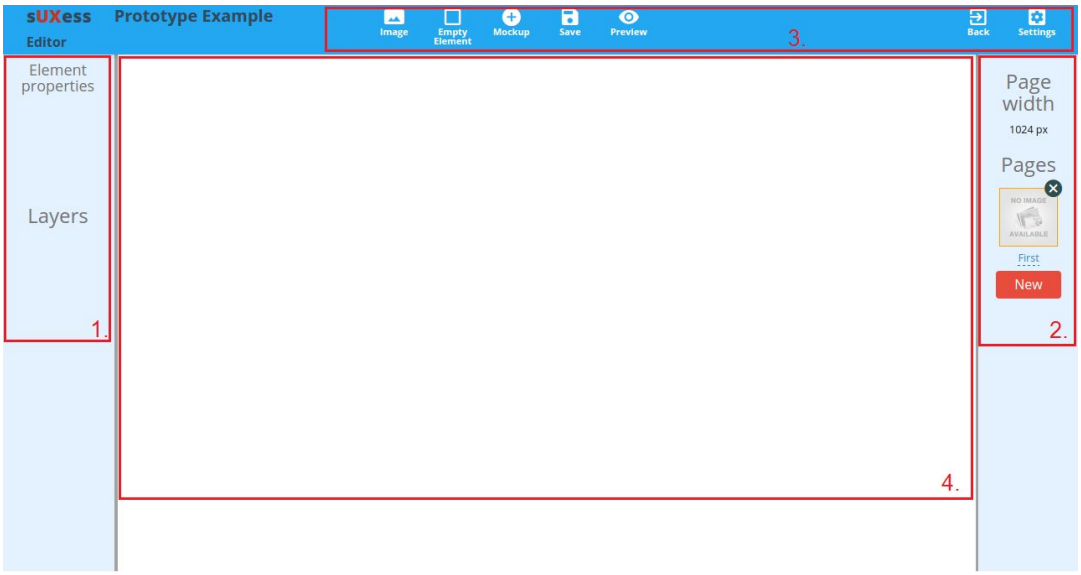
F Používateľská príručka pre editor

Používateľská príručka - Editor

- Opis editora
- Pridanie nového image elementu
- Pridanie nového prázdneho elementu
- Manipulácia elementov
    - Zmena názvu elementu
    - Pridanie prelinkovania na iný mockup
    - Simulovanie prelinkovania
    - Odstránenie prelinkovania
- Odstránenie elementu
- Pridanie nového mockupu
    - Zmena názvu mockupu
    - Odstránenie mockupu
- Uloženie prototypu
- Nastavenie šírky stránky
- Spustenie prehliadky prototypu
- Zavretie editora

Opis editora

Editor slúži na manuálne vytváranie prototypov testovaných stránok alebo aplikácií. Je možné v ňom vytvárať jednotlivé podstránky reprezentované mockupmi. Tieto mockupy potom obsahujú elementy danej podstránky, ktorým je možné nastaviť možnosť vystupovať v roly linku.



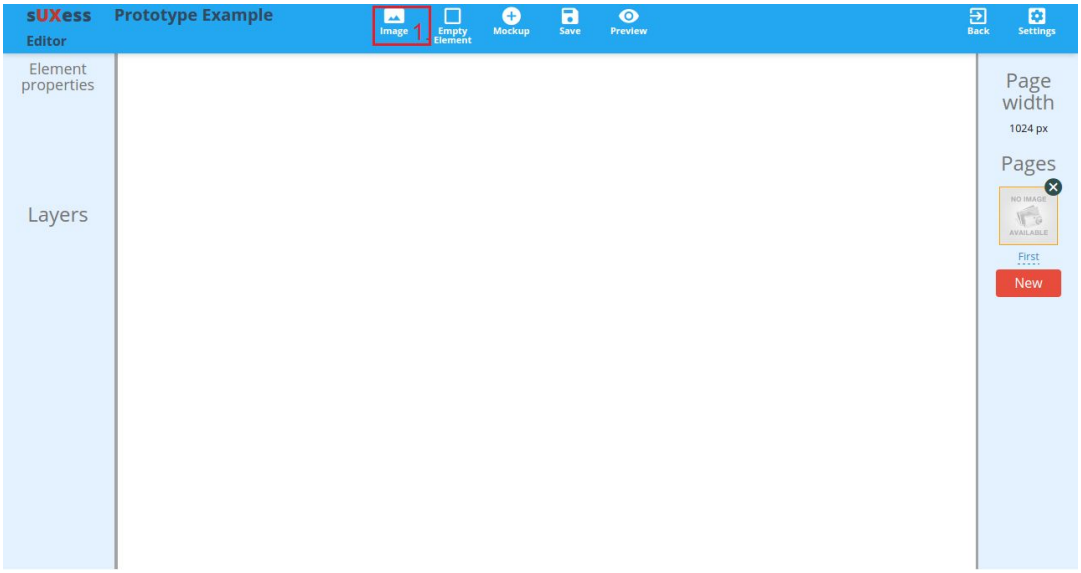
Obrázok 1

Samotný editor je rozdelený do niekoľkých častí:

1. Ľavá časť obsahuje informačný panel, na ktorej sú zobrazené údaje o elementoch práve vybraného mockupu. Časť “Layers”obsahuje zoznam všetkých elementov, ktoré obsahuje vybraný mockup. Po označení elementu sa v časti “Element Properties” zobrazia údaje o tomto elemente. Táto časť je zobrazená na obrázku 1 v časti 1.
2. Pravá časť obsahuje zoznam mockupov prototypu. Rovnako obsahuje informáciu o šírke všetkých mockupov daného prototypu. Táto časť je zobrazená na obrázku 1 v časti 2.
3. Horný ovládací panel obsahuje zoznam nástrojov, ktoré je možné používať počas tvorby mockupu a tiež obsahuje možnosť na prehliadanie celého prototypu. Táto časť je zobrazená na obrázku 1 v časti 3.
4. Posledná časť, ktorá vypĺňa väčšinu obrazovky je v strede a predstavuje obsah samotného mockupu. Sem sú umiestňované jednotlivé elementy. Táto časť je zobrazená na obrázku 1 v časti 4.

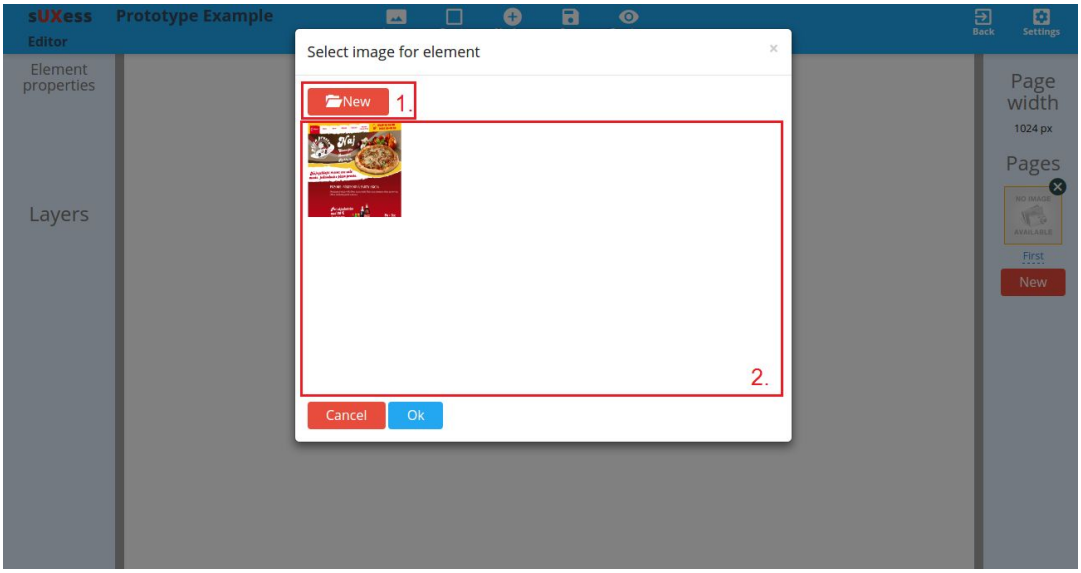
Pridanie nového image elementu

Pre pridanie nového elementu, ktorý obsahuje obrázok je potrebné kliknúť na ikonku “Image” na paneli nástrojov. Na obrázku 2 je táto ikonka zobrazená časťou 1.



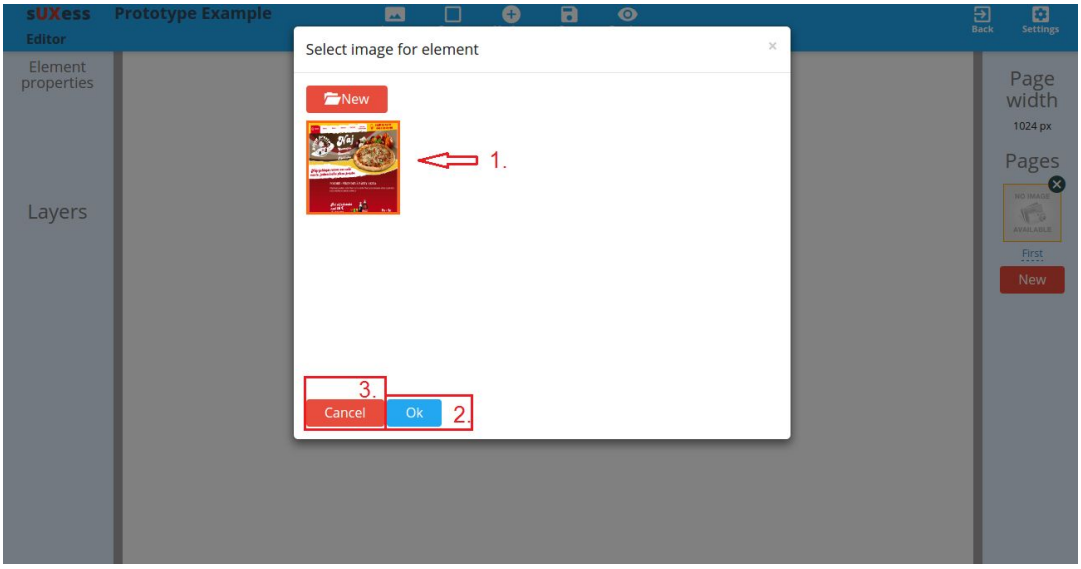
Obrázok 2

Po kliknutí na tento nástroj sa zobrazí ponuka na výber obrázka, ktorý má element obsahovať. Ponuka obsahuje všetky doteraz nahrané súbory používateľa a rovnako poskytuje možnosť nahrať nový súbor. Okno pre výber obrázka je vidieť na obrázku 3, kde časť 1 predstavuje tlačidlo pre pridanie nového obrázku a časť 2 predstavuje ponuku už nahraných obrázkov používateľom.



Obrázok 3

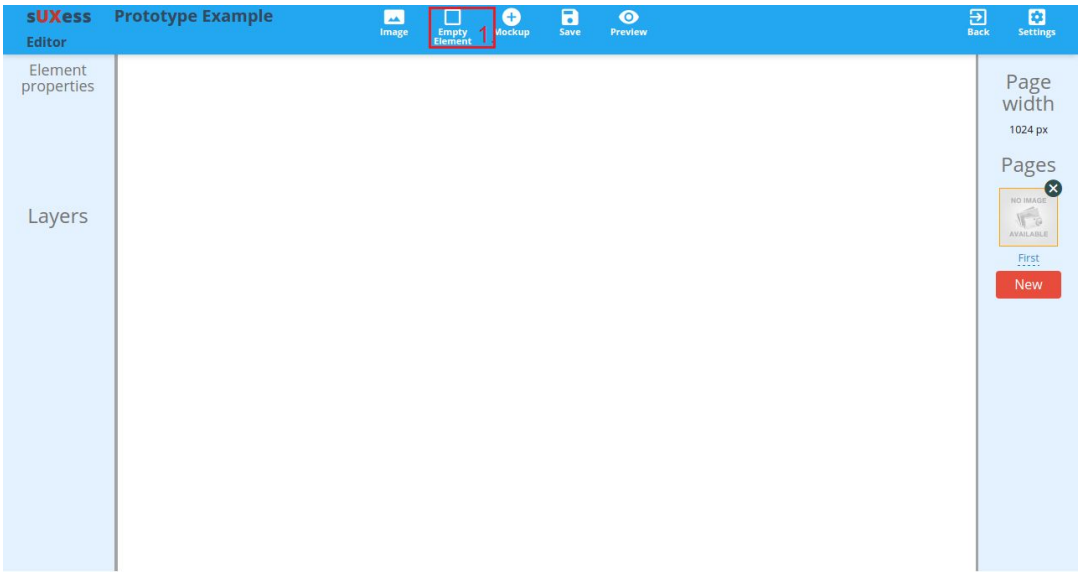
Pre priradenie nového obrázka vytváranému elementu je potrebné z ponuky vybrať obrázok kliknutím myši. Na obrázku 4 je zobrazený výber obrázka s už vybraným súborom v časti 1. Potvrdenie výberu a dokončenie vytvorenia elementu je vykonané kliknutím na tlačidlo “Ok”, ktoré je v časti 2. Zrušenie tvorby elementu je vykonané kliknutím na tlačidlo “Cancel” v časti 3.



Obrázok 4

Pridanie nového prázdneho elementu

Pre vytvorenie prázdneho elementu slúži nástroj umiestnený na ovládacom paneli nástrojov označený ako “Empty Element”. Ikonka nástroja je zobrazená na obrázku 5 v časti 1.



Obrázok 5

Na rozdiel od elementu obsahujúceho obrázok nie je potrebné vyberať obsah prázdneho elementu a preto sa tento element priamo vytvorí. Keďže prázdny element neobsahuje žiadny obsah, je pre viditeľnosť vyplnený modrou farbou ako je vidno na obrázku 6.

Obrázok 6

Manipulácia elementov

Presúvanie elementov je vykonávané jednoduchým označením elementu, podržaním ľavého tlačidla myši a následným potiahnutím na požadovanú pozíciu. Ďalšími možnosťami manipulácie elementov sú zmena názvu elementu, pridanie prelinkovania, simulovanie prelinkovania a odstránenie prelinkovania.

Zmena názvu elementu

Pri pridávaní elementov sú elementy automaticky pomenovávané ako “element” a jeho poradové číslo v poradí ako bol vytvorený. Unikátne názvy elementov sú dôležité pre výber oblasti záujmu pri vytváraní Taskov pre jednotlivé Testy. Zmeniť aktuálny názov elementu je možné spraviť v aktuálne vybranom mockupe a to dvojitým kliknutím na názov elementu v zozname elementov označenom ako “Layers”. Na obrázku 7 je zobrazené editovateľné okno s názvom elementu, ktoré sa objaví po dvojitom kliknutí. Časť 1 vyjadruje element v zozname elementov, na ktorý je potrebné dvakrát kliknúť.

Obrázok 7

Pre zmenu názvu elementu je potrebné zmeniť meno, ktoré je vidno v časti 2 a pre potvrdenie tejto zmeny stlačiť tlačidlo zobrazené v časti 3. Pre zrušenie zmeny názvu elementu stačí stlačiť tlačidlo v časti 4.
Pri pridaní viacerých elementov môže nastať situácia, že sa niektoré elementy navzájom prekrývajú. Elementy sú v zozname elementov “Layers” usporiadané podľa toho, v akom poradí sa navzájom prekrývajú. Toto poradie je možné ľubovoľne meniť kliknutím a podržaním ľavého tlačidla myši na element v zozname elementov a následné potiahnutie vybraného elementu na požadovanú úroveň.

Pridanie prelinkovania na iný mockup

Pre pohyb medzi jednotlivými mockupmi vo funkčnom prototype je možné použiť nástroj, ktorý umožňuje pridávať elementom odkazy na iný mockup. Tento nástroj sa viaže na konkrétny element a je vyvolaný kliknutím pravého tlačidla myši na element. Po stlačení pravého tlačidla myši sa objaví menu tak ako je to na obrázku 8 v časti 1.

Obrázok 8

Toto menu obsahuje tri možnosti, kde prvá je názov vybraného elementu. Druhá možnosť je pre pridanie odkazu na iný mockup. Pre zobrazenie ponuky mockupov, na ktoré je možné použiť prelinkovanie je potrebné ukázať myšou na možnosť “Add link”. Potom sa zobrazí ponuka ako je to na obrázku 9 .

Obrázok 9

V ukážkovom príklade je vytvorený iba jeden mockup, preto zoznam mockupov pre prelinkovanie obsahuje len jednu položku a tou je mockup s názvom “First” v časti 1. Informácia o aktuálnom elemente a jeho prelinkovaní je zobrazená na informačnom paneli, ktorý obsahuje “Element properties”.

Simulovanie prelinkovania

Ak je potrebné simulovať element ako link, ale zároveň nie je potrebné aby naozaj niekam odkazoval, je možné použiť variantu prelinkovania “Blank page”. Táto možnosť simuluje zobrazenie elementu ako link, čo znamená, že po nájdení myšou na element sa zobrazí kurzor ruky, ale po kliknutí nevykoná žiadnu akciu. K nastaveniu tejto možnosti je možné sa dostať rovnako ako k pridaniu odkazu na iný mockup, ale namiesto mockupu vyberieme možnosť “Blank page”, ktorá je zobrazená v časti 2 na obrázku 9 .

Odstránenie prelinkovania

Ak už nie je naďalej žiadúce, aby element odkazoval na iný mockup, je možné toto prelinkovanie odstrániť variantou “Remove link”, ku ktorej je možné sa dostať rovnako ako pri pridávaní odkazu na iný mockup, ale vyberieme možnosť “Remove link” v časti 3 na obrázku 9 .

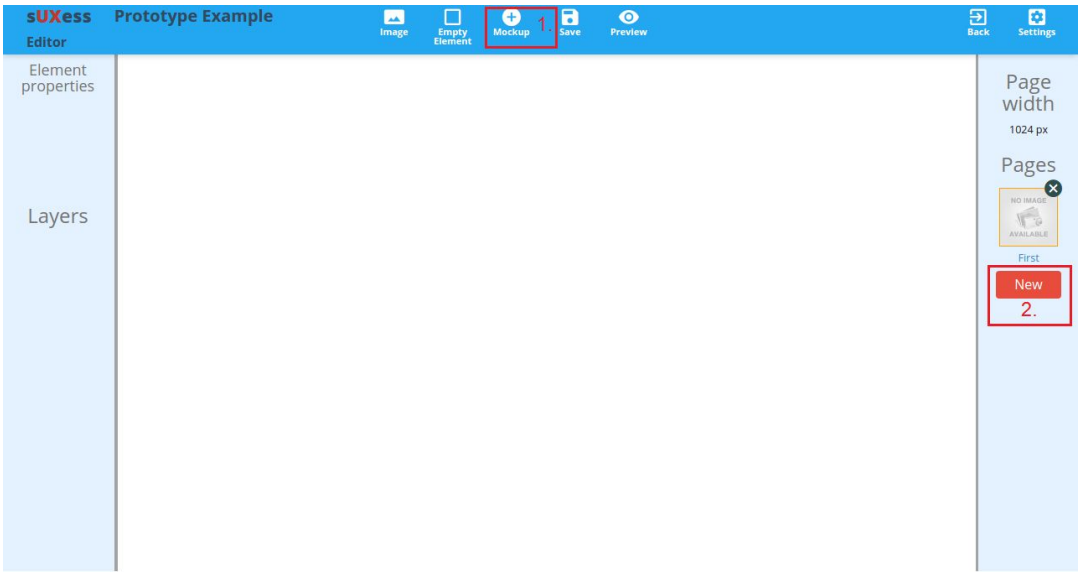
Odstránenie elementu

V prípade, že je potrebné nejaký element odstrániť, je možné túto akciu vykonať v dvoch krokoch. Kliknutím pravého tlačidla myši na element, ktorý chceme odstrániť vyvoláme ponuku nástroja. V zobrazenej ponuke vyberieme možnosť “Delete” ako je to na obrázku 10 v časti 1.

Obrázok 10

Pridanie nového mockupu

Ak chceme do prototypu pridať nový mockup máme na výber z dvoch možností. Na obrázku 11 sú vyznačené oblasti 1 a 2, ktoré umožňujú vytvorenie nového mockupu. Pre aktiváciu tohoto nástroja stačí kliknúť na jedno z týchto dvoch tlačidiel.



Obrázok 11

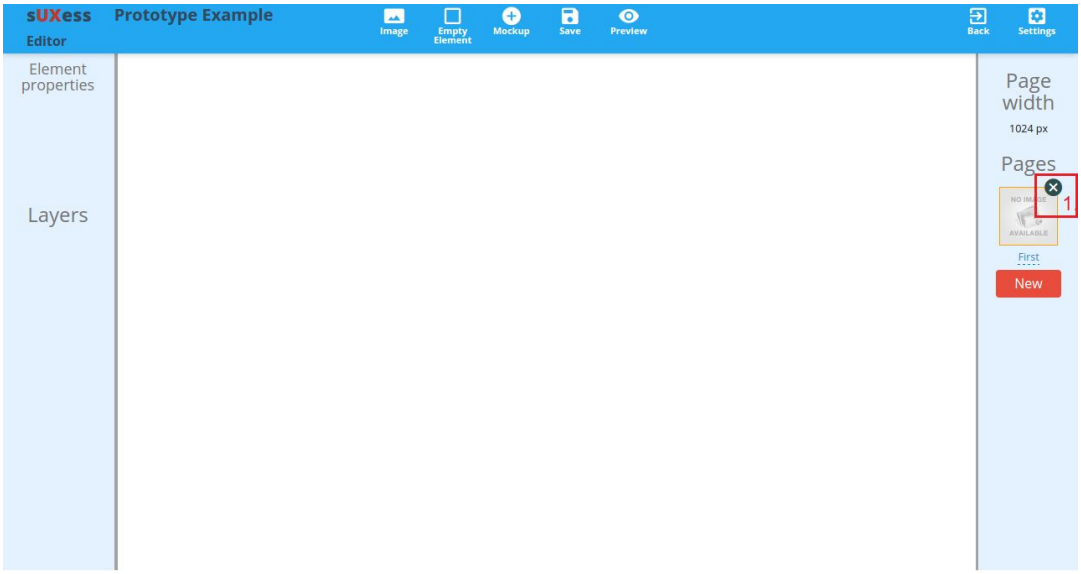
Zmena názvu mockupu

Pre vytváranie prelinkovaní medzi mockupmi je dobré mať definované unikátne mená mockupov. Premenovať mockup je možné podobne ako v prípade premenovania elementu. Rozdiel je, že v prípade mockupov je potrebné kliknúť na názov mockupu jedenkrát ako je to na obrázku 12 v oblasti 1. Pre editáciu názvu mockupu slúži textové pole v oblasti 2. Pre potvrdenie zmeny názvu mockupu slúži tlačidlo v oblasti 3 a pre zrušenie zmeny názvu tlačidlo v oblasti 4.

Obrázok 12

Odstránenie mockupu

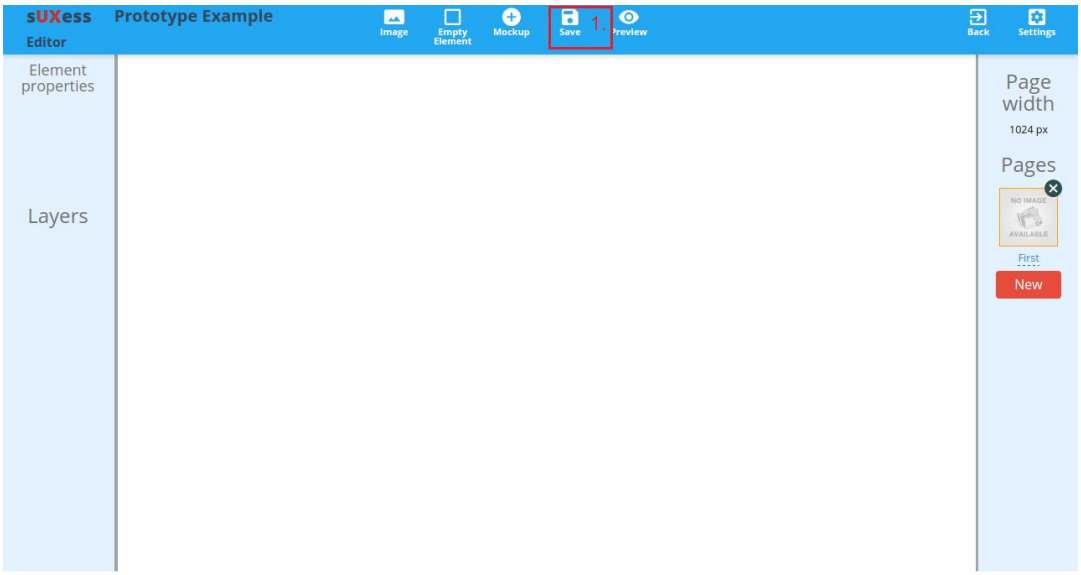
V prípade, že chceme odstrániť nejaký mockup, je možné to vykonať v jednom kroku a to kliknutím na tlačidlo v oblasti 1 na obrázku 13 .



Obrázok 13

Uloženie prototypu

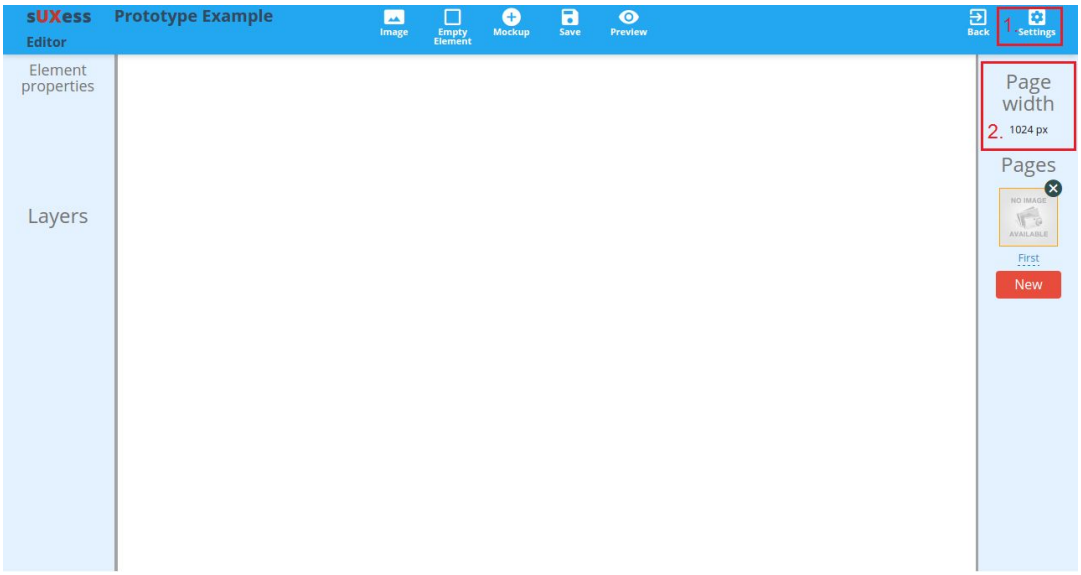
Po upravení jedného alebo viacerých mockupov je potrebné uložiť vykonané zmeny, v opačnom prípade sa pri znovu načítaní prototypu vykonané zmeny neprejavia. Uložiť prototyp je možné kliknutím na tlačidlo “Save” v ovládacom paneli nástrojov ako je to na obrázku 14 v časti 1.



Obrázok 14

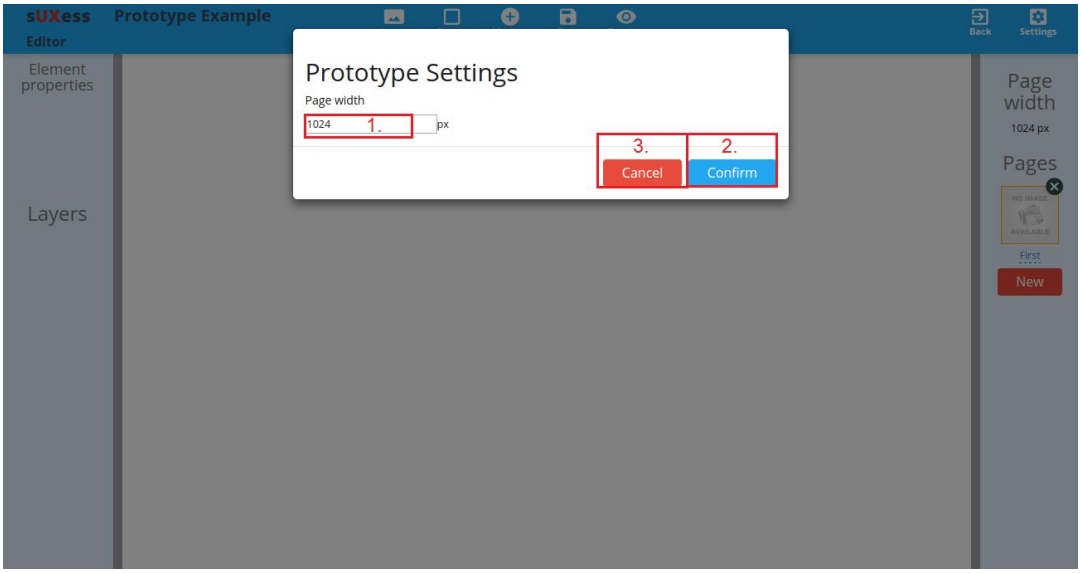
Nastavenie šírky stránky

Pre fixné nastavenie šírky každej podstránky prototypu z dôvodu požadovaného rozlíšenia je možné toto nastavenie zmeniť kliknutím na tlačidlo “Settings” v ovládacom paneli nástrojov tak ako je to na obrázku 15 v oblasti 1.



Obrázok 15

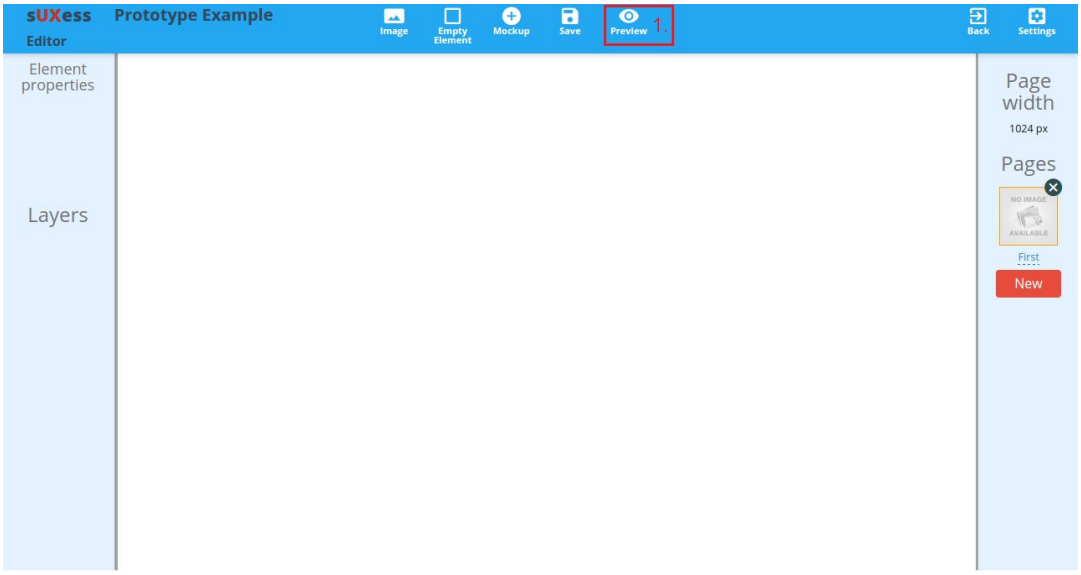
Súčasné nastavenie šírky stránky je možné vidieť na paneli vpravo v časti “Page Width” zobrazenej v oblasti 2.
Po stlačení tlačidla “Settings” sa objaví okno, v ktorom je možné toto nastavenie zmeniť. Na obrázku 16 je zobrazené vyskakovacie okno, kde v oblasti 1 je možné zmeniť hodnotu šírky okna, ktorá je udávaná v pixeloch. Potvrdenie zmeny sa vykoná stlačením tlačidla “Confirm” v oblasti 2 a zrušenie vykonaných zmien stlačením tlačidla “Cancel” v oblasti 3.



Obrázok 16

Spustenie prehliadky prototypu

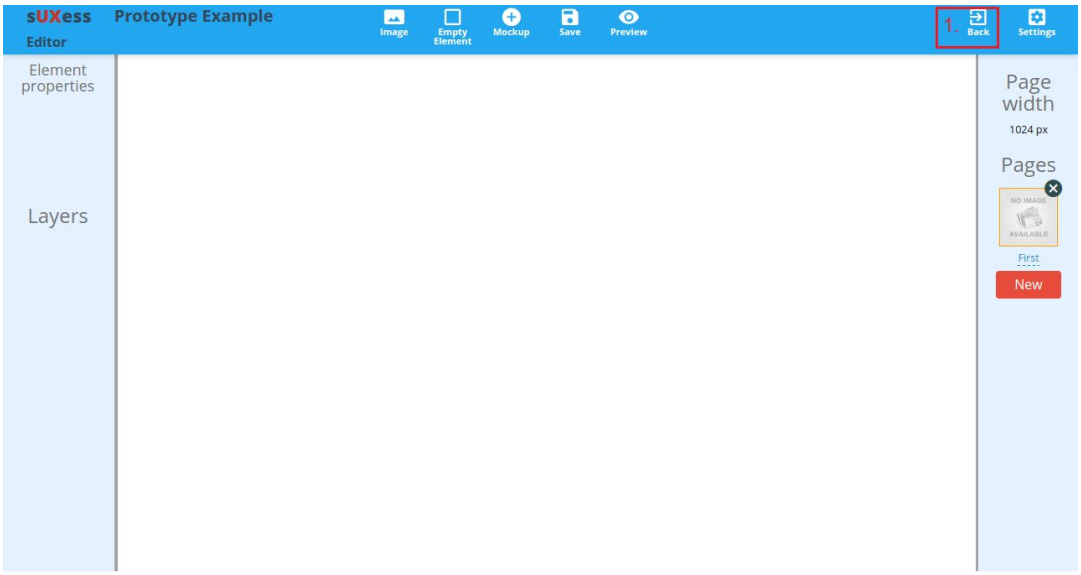
Pre prehliadanie prototypu tak ako ho uvidí tester je možné použiť nástroj na ovládacom paneli nástrojov s názvom “Preview”, ktorý simuluje vytvorený prototyp vrátane prelinkovania. Tlačidlo na spustenie tejto prehliadky je zobrazené na obrázku 17 v oblasti 1.



Obrázok 17

Zavretie editora

Po ukončení práce na prototype a uložení všetkých vykonaných zmien je možné ukončiť editor jediným kliknutím na tlačidlo “Back”, ktoré vráti tvorcu prototypu späť na výber prototypov. Tlačidlo “Back” sa nachádza na ovládacom paneli naštrojov tak, ako je to na obrázku 18 v oblasti 1.



Obrázok 18