

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	2
1.1 Úvod	2
1.2 Role členov tímu a podiel práce	2
1.2.1 Role členov tímu	2
1.2.2 Podiel na dokumentácii riadenia projektu	3
1.2.3 Podiel na dokumentácii k inžinierskemu dielu	3
1.3 Aplikácie manažmentov	5
1.3.1 Procesy manažmentu projektu	5
1.3.2 Procesy manažmentu softvéru	6
1.3.3 Procesy vývoja a prevádzky softvéru	6
1.4 Sumarizácie šprintov	7
1.4.1 1. šprint	7
1.4.2 2. šprint	7
1.5 Používané metodiky	7
1.5.1 Metodika verziovania	8
1.5.2 Metodika revízie kódu	8
1.5.3 Metodika kontinuálnej integrácie	8
1.5.4 Metodika pre správu riadenia projektu a požiadaviek	8
1.5.5 Metodika pre písanie kódu v JavaScripte	8
1.5.6 Metodika pre písanie kódu v Ruby on Rails	8
1.5.7 Metodiky pre testovanie aplikácie	8
1.6 Globálna retrospektíva	8
2 Inžinierske dielo	10
2.1 Úvod	10
2.2 Ciele projektu	10
2.2.1 MVP pre zimný semester	11
2.3 Architektúra	13
2.3.1 Frontend	14
2.3.2 Backend	15
2.3.3 Komponenty systému	16
2.3.4 Dátový model	17
2.3.5 Tímový server	18
A Zoznam kompetencií tímu	19
B Metodiky	20
C Export evidencie úloh	45

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 zosumari-zované 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 Role členov tímu a podiel práce

1.2.1 Role 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

1.2.2 Podiel na dokumentácii riadenia projektu

Podiel práce členov tímov na dokumentácii k riadeniu projektu je nasledovný:

Jakub Vrba

- 1.5 Používané metodiky
- Metodiky

Monika Sanyová

- Zoznam kompetencií tímu

Patrik Gajdošík

- 1.1 Úvod
- 1.2 Role členov tímu a podiel práce
- Metodika kontinuálnej integrácie

Peter Dubec

- Zoznam kompetencií tímu

Roman Roba

- 1.4 Sumarizácie šprintov
- 1.6 Globálna retrospektíva
- Export evidencie úloh

Róbert Cuprik

- Zoznam kompetencií tímu

Tomáš Žigo

- 1.3 Aplikácie manažmentov

1.2.3 Podiel na dokumentácii k inžinierskemu dielu

Podiel práce členov tímov na dokumentácii k inžinierskemu dielu je nasledovný:

Jakub Vrba

- Angular VC model

Monika Sanyová

- Dátový model

- Rails API

Patrik Gajdošík

- 2.3.5 Tímový server

Peter Dubec

- Ciele projektu
- Rails API

Roman Roba

- 2.1 Úvod

Róbert Cuprik

- Angular VC model
- Diagram komponentov

Tomáš Žigo

- Vysoká úroveň

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 rozdelený 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 pridelené 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 vyjadří 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 funkcionalitu 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. Informácie o tom ako vykonávať revíziu svojho kódu 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 funkcionality produktu definoval na začiatku vývoja. Nasledujúce požiadavky sú upravované a dopĺňané na tímových stretnutiach na základe tímových diskusií.

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 uvedené 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 funkcionality. Každý člen je zodpovedný za testovanie svojho kódu. Bližšie informácie o tom ako otestovať svoju funkcionality v backende alebo frontende sú uvedené 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

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.

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.

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 funkcionality 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.

Spolu s nedokončenými úlohami sme vytvorili nové úlohy pre nasledujúci šprint. 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.

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 z AIS (LDAP), sme sa rozhodli prestať venovať čas tomuto problému aby sme sa mohli venovať dôležitejším veciam.

Po ukončení šprintu sme vytvorili nové úlohy pre nasledujúci šprint.

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 branchy, kedy vytvárať nové branchy 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 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

Po ukončení jednotlivých šprintov, sme si vždy daný šprint vyhodnotili a vytvorili k nemu retrospektívu. V tejto retrospektíve sme vždy poukázali na procesy, ktoré sa nám overili a chceme v nich ďalej pokračovať. Ďalej sme poukázali na procesy ktorým sa chceme v budúcnosti vyhnúť a na procesy, ktoré chceme začať dodržiavať.

Vyhnúť sa v budúcnosti:

- Nevyvíjať nad „cudzou“ vetvou.
- Nenechávať si všetko na poslednú chvíľu.

Pokračovať v tom ďalej:

- Dodržiavať konvencie.
- 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*.
- Začať používať Jiru počas práce na taskoch.

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, kde sa snažíme vytvoriť platformu pre online UX testovanie spolu so sadou základných UX nástrojov na testovanie webových stránok.

V kapitole 2.2 opisujeme ciele projektu pre zimný semester. Tieto ciele sa zameriavajú na minimálnu funkcionálnosť, ktorú sme si zaumienili implementovať ešte počas zimného semestra tak, aby sme mali funkčný prototyp.

Kapitola 2.3 sa sústreďuje na architektúru nášho systému, kde poskytuje 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:

- navrhnuť a implementovať platformu pre online UX testovanie
- navrhnuť 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 funkcionality, ktorá v doterajších nástrojoch pre online UX testovanie nie je dostupná.

Účastníci budú k testu pridelení 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 zbierať tak veľké množstvo dát.

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

2.2.1 MVP pre zimný semester

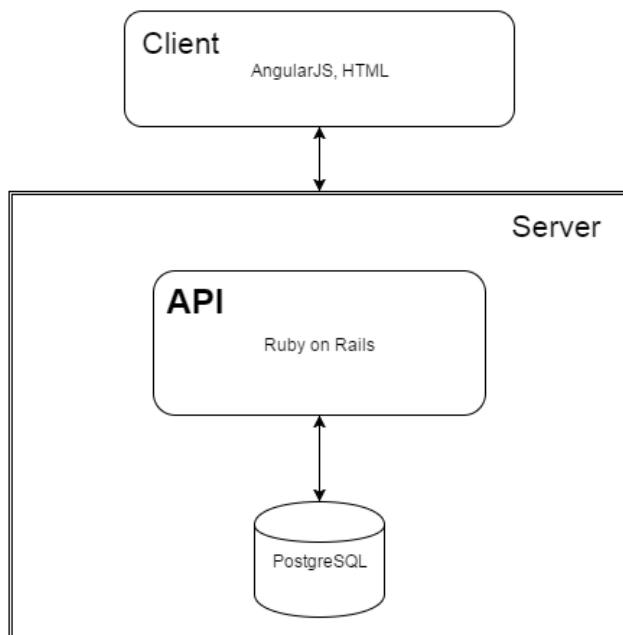
V rámci našej práce na projekte počas zimného semestra sme si ako cieľ stanovili vytvoriť počas zimného semestra funkčný prototyp, ktorý bude nasadený online a bude spĺňať nasledovnú funkcionality:

- Možnosť registrácie používateľov
- Zadávatel 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 metrik
 - štatistika splnenia / nesplnenia úlohy
 - časy trvania úloh
 - metriky nad oblasťami záujmu
 - čas do prvého kliknutia
 - čas do prvej návštevy
 - počet kliknutí

2.3 Architektúra

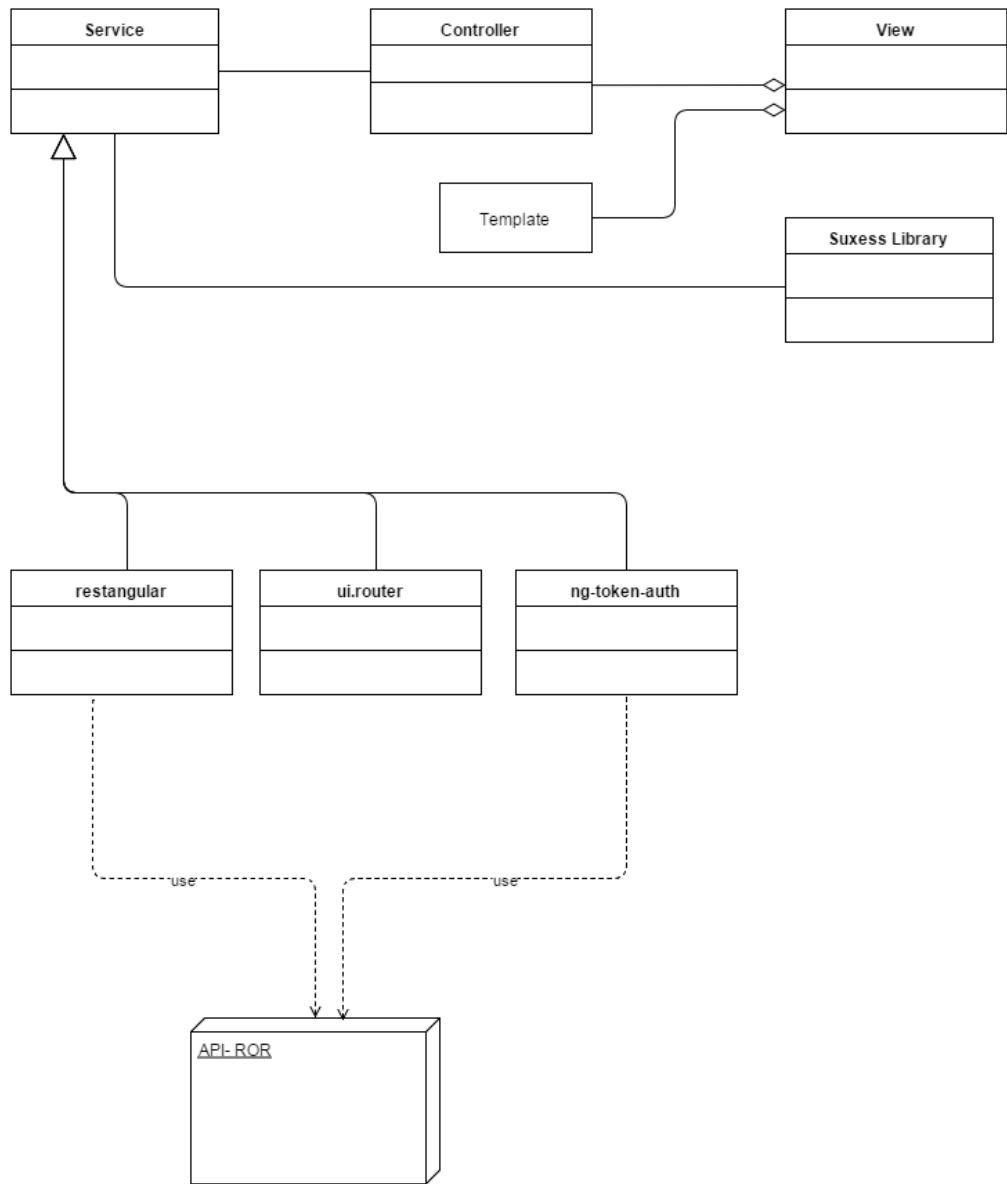
Architektonický štýl klient–server vyjadruje architektúru, v ktorej má aplikácia dve časti – klientskú časť a serverovú časť. Klientská časť je prezentovaná používateľovi a všetky dáta žiada od servera, kým serverová časť, ktorá je reprezentovaná ako API, vykonáva aplikáčnú logiku a pristupuje k databáze. 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.



Obr. 1: Diagram architektúry.

2.3.1 Frontend

Na obr. č. 2 je znázornený diagram architektúry frontedu. Využívame knižnicu Angularjs a tomu je aj prispôbená architektúra. Zobrazovaná stránka je reprezentovaná triedov View, ktorá v sebe agreguje príslušný controller a html template. 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.

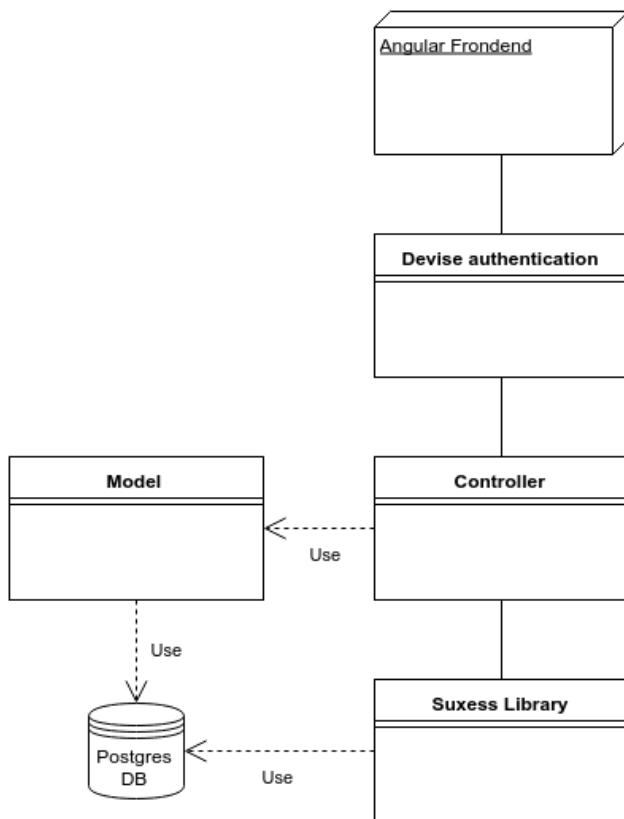


Obr. 2: Diagram architektúry frontedu.

2.3.2 Backend

Na obr. č. 3 je zobrazený diagram architektúry backendu. 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.



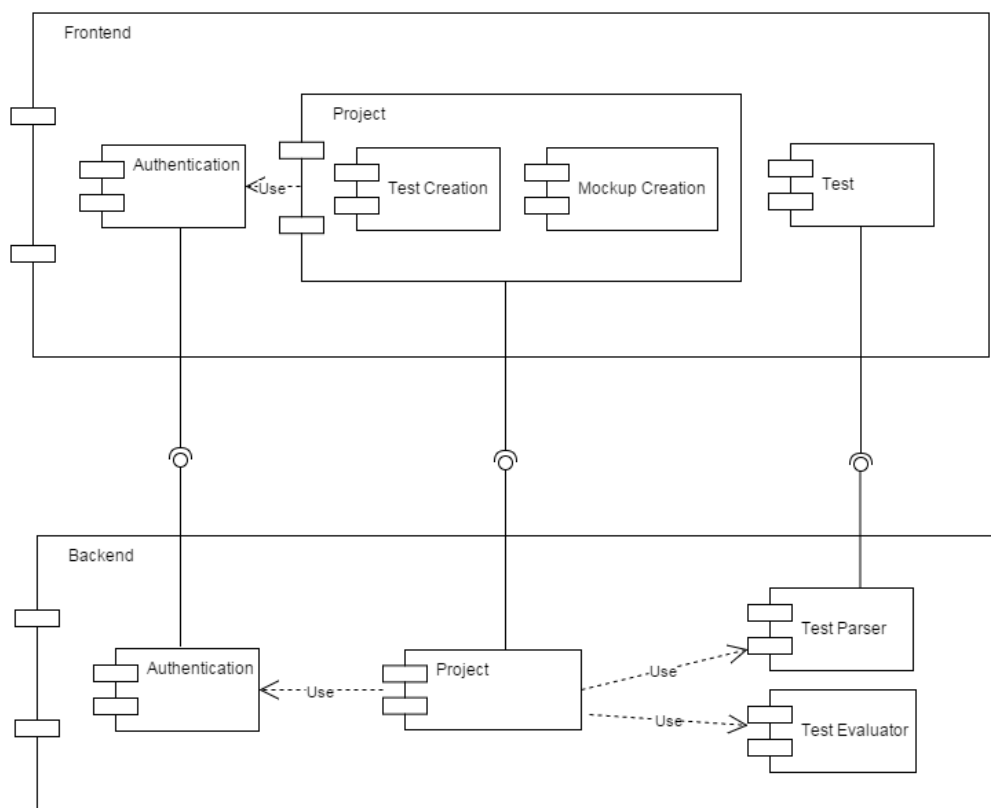
Obr. 3: Diagram architektúry backendu.

2.3.3 Komponenty systému

Na diagrame na obr. č. 4 je znázornený diagram komponentov systému. Opisuje komponenty, ktoré sme doteraz identifikovali a ich komunikáciu.

Na frontende existujú tri komponenty. *Autentifikácia*, ktorá komunikuje s rovnomeným 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. Komponent *Test* slúži na testovanie mockupov používateľmi.

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.



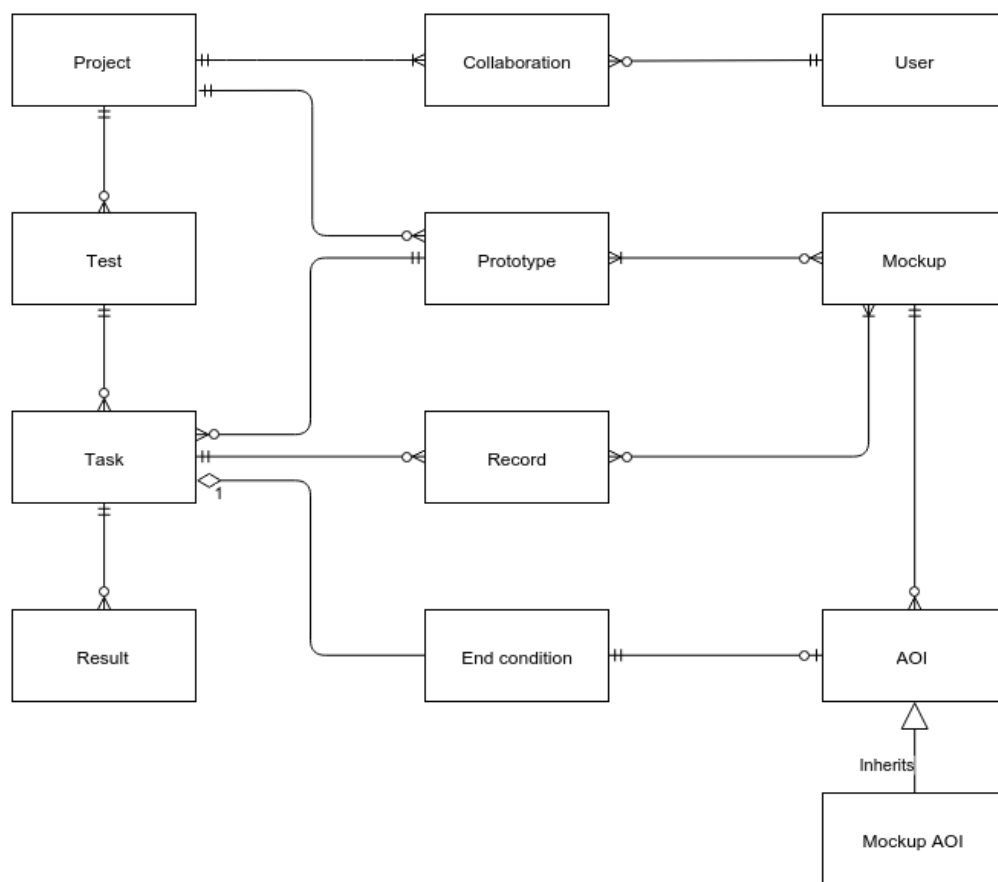
Obr. 4: Diagram komponentov systému.

2.3.4 Dátový model

Na obr. č. 5 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*. Projekt predstavuje testovaný produkt. V rámci projektu môže používateľ vytvárať testy (entita *Test*), ktoré sa skladajú z menších úloh (entita *Task*).

V rámci projektu si môže používateľ nahrávať rôzne obrázky/screenshoty (entita *Mockup*), nad ktorými môže definovať rozličné oblasti záujmu *AOI*. 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 (*End condition*), zväčša viazanú na konkrétnu oblasť záujmu.

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*.

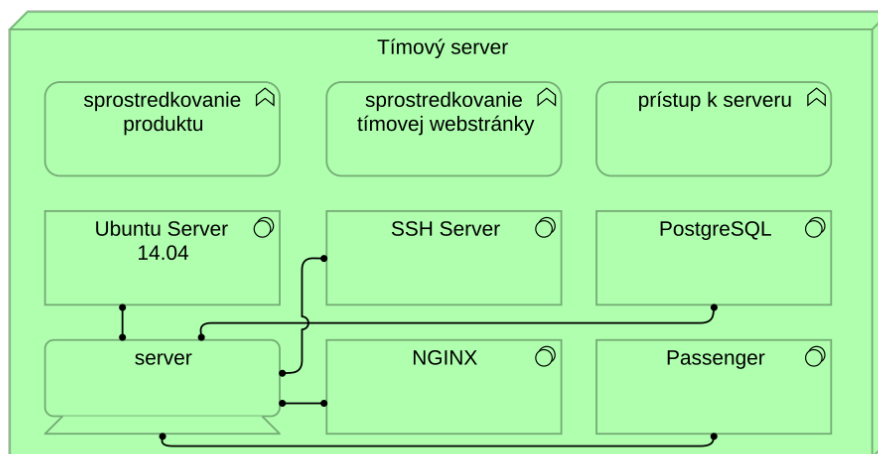


Obr. 5: Diagram dátového modelu.

2.3.5 Tímový server

Prevádzku našej tímovej stránky, ako aj nášho produktu, zabezpečuje tímový server. Ako je možné vidieť na obrázku č. 6. 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.



Obr. 6: Intraštruktúrny pohľad na tímový server.

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.

A Zoznam kompetencií tímu

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, Android, Java
Tomáš Žigo	HTML, Java, C, PostgreSQL

Tabuľka 1: Zoznam kompetencií členov tímu

B Metodiky

Metodika verziovania

1. Naklonovanie celého projektu k sebe

Na githube je [https/sshlink](https://sshlink) na clone projektu, treba ho skopírovať a do konzoly dať príkaz:
`git clone <ten skopírovaný link>`

2. Vytvorenie novej branche z develop-u

```
git checkoutdevelop
```

```
git pullorigindevelop
```

```
git checkout -b nazov-novej-branche
```

Teraz som už v novej branchi.

3. Naklonovanie existujúcej branche z githubu

```
git fetch
```

```
git checkout -b nameorigin/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. Pullrequest

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

```
git add .
```

```
git commit -m "zmysluplna sprava o poslednychzmenach v mojej feature branchi"
```

```
git checkoutdevelop
```

```
git pullorigindevelop # teraz mám najnovšiu verziu developu
```

```
git checkout feature/moja-feature
```

```
git merge --no-ffdevelop
```

Ak nastali konflikty, vyriešim ich, uložíam zmeny novým commitom, ktorého názov bude napr. `resolvemergeconflicts` a dokončím merge znovu príkazom `git merge --no-ffdevelop`.

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 pushorigin feature/moja-feature

Priamo na githube (uz nepoužívam konzolu) vytvorím PR na mergnutie 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.

Metodika revízie kódu

Použi nasledujúce gemy:

* brakeman

* rails_best_practices -f html .

* rubycritic

Pre revíziu kódu v Javacsripte použi:

* grunt jshint

Príkaz na inštaláciu gemu:

```
gem install<name>
```

Nainštaluj jshint rovnakým spôsobom ako ostatné knižnice, čiže pomocou gruntu.

Spusti dané knižnice na svojom kóde a uprav chyby, ktoré nastali pri písaní.

Metodika kontinuálnej integrácie

Obsah:

- Wercker
- Testovanie
- Nasadenie
- Závislosti
- Prístup k serveru

Wercker

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

```
yaml

# Docker kontajner, v ktorom sa uskutočnia všetky kroky.

box: mwallasch/docker-ruby-node

# Toto je buildpipeline. V nej prebiehajú všetky kroky počas testovania.

# Viac sa môžete dozvedieť 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: developmentproductiondeployment

    - script:

name: rspec

code: bundleexecrspec

# Toto je deploypipeline. Spúšťa sa v prípade spustenia nasadenia.

deploy:

    # codereview vymedzuje kroky špecifické iba pre určitý druh nasadenia.

codereview:

    - script:

name: installrubycritic

code: gem installrubycritic --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

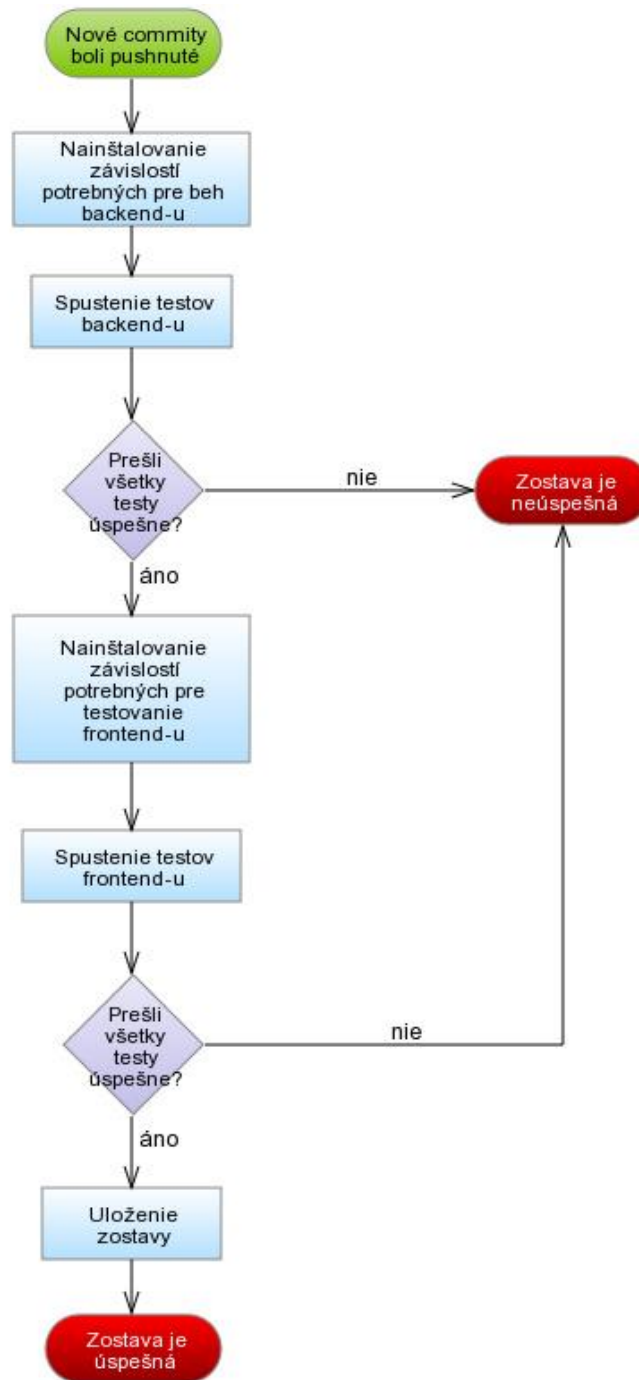
Ak ktorýkoľvek krok zlyhá, zvyšné kroky (okrem after-steps) sa už nevykonajú.

Testovanie

Testovanie prebieha v dvoch častiach:

- testovanie backendu - spúšťajú sa rspec testy;
- testovanie frontendu - spúšťajú sa jasmine testy.

Proces testovania prebieha v docker kontajneri a spúšťa ho služba Wercker. Sú vykonávané viaceré kroky. Ako je možné vidieť na obrázku X, po tom, ako sa zaznamenajú nové zmeny v zdrojových kódach, sú tieto kódy stiahnuté. Pre beh je potrebné taktiež nainštalovať závislosti a následne sa spúšťajú testy, najskôr backend a potom frontend. Ak v ktorejkoľvek fáze testy zlyhajú, proces je prerušený a takáto zostava je označená za neúspešnú.

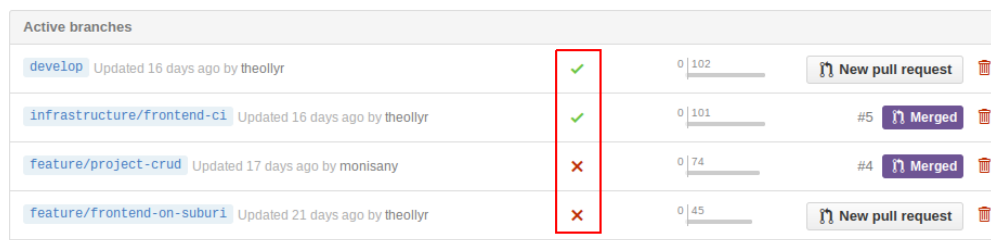


Obrázok č. 1: Proces testovania novej verzie zdrojových kódov

Ak nechceme, nie je potrebné, aby sa push-nuté commity otestovali (napríklad

došlo iba k zmene v README), tak hocikam do názvu commitu dáme [skipci] alebo [ciskip].

Wercker sa automaticky integruje do rozhrania GitHub-u a ako je možné vidieť na obrázku 2, sú tam zobrazené stavy jednotlivých zostáv v závislosti od vetvy. V prípade, že zostava vetvy zlyháva, je potrebné opraviť chyby, ktoré spôsobujú neúspešné testy.



Branch	Updated	By	Status	Commits	Build Status	Actions
develop	Updated 16 days ago	by theollyr	✓	0	102	New pull request
infrastructure/frontend-c1	Updated 16 days ago	by theollyr	✓	0	101	#5 Merged
feature/project-crud	Updated 17 days ago	by monisany	✗	0	74	#4 Merged
feature/frontend-on-subur1	Updated 21 days ago	by theollyr	✗	0	45	New pull request

Obrázok č. 2: Zobrazenie stavu zostavy v rozhraní GitHub-u

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(<http://capistranorb.com/documentation/getting-started/configuration/>)

sa nachádza v Capfile, config/deploy.rb a config/deploy/production.rb.

Konfigurácia Grunt-u(<http://gruntjs.com/configuring-tasks>) 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.

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 pridať 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
```

Metodika pre správu riadenia projektu a požiadaviek

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

JIRA

- odkaz na jiru(<https://jira.fiit.stuba.sk/>)

Metodika pre písanie kódu v JavaScripte

Angular.js

Folderstructure

Používaj už predvytvorenú súborovú štruktúru:

/angular-app

/controllers

/directives

/filters

/modules

/services

/templates

Podpriechinky vytváraj tak, aby spájali súbory, ktoré spolu logicky súvisia dokopy, príklad:

/angular-app

/controllers

/project

projectController.js

projectDetailController.js

/directives

/project

/user

/filters

/modules

/services

/templates

Konvencia pomenovania

Používaj v názve súboru typ angulárovského objektu:

projectController.js

userLoginDirective.js

Používaj camelCase pre JavaScriptové súbory:

projectController.js

Súbory by mali mať rovnaké meno ako angulárovský objekt, príklad:

projectController.js

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

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

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

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

Pomenuj html súbory podľa ich prislúchajúceho angulárovského objektu, príklad:

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

project-create-directive.html

SUXESS knižnica

Používaj globálnu premennú SUXESS na definovanie triedy

```
SUXESS.SecureStateCheck = function () {...}
```

```
var secureCheck = new SUXESS.SecureStateCheck(..);
```

Nevytváraj nový namespace, príklad:

```
SUCCESS.namespace.SecureStateCheck
```

Spájaj triedy do priečinkov:

```
/suxess
```

```
  /utills
```

```
secureStateCheck.js
```

```
suxess.js
```

Štýl kódu

Používaj pravidlá zadané v airbnb(<https://github.com/airbnb/javascript>).

V stringoch používaj ' 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 skrátené na jedno písmeno, začínajúc od písmena i.

```
variableNamesLikeThis
```

```
functionNamesLikeThis
```

```
ClassNamesLikeThis
```

```
methodNamesLikeThis
```

```
ConstantsLikeThis
```

Užitočné odkazy:

Airbnb(<https://github.com/airbnb/javascript>)

Dokumentácia a komentovanie kódu

Na dokumentovanie metód používaj jsdoc, kde do popisu napíš, čo robí daná metóda, aké sú jej vstupné a výstupné parametre. Snaž sa písať kód tak, aby neboli potrebné komentáre. Môžu nastať, ale prípady, kde je to treba a vtedy sa snaž napísať komentár, ktorý popíše daný kus kódu.

Metodika pre písanie kódu v 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 Camelcase (napr. ClassName), všetko ostatné Snakecase (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

```
defsplit_message()
```

```
  @msg.split() if (@msg.size > 20)
```

```
end
```

Do this

```
defsplit_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
```

Useunlessstatement

```
unless @read_only
```

```
end
```

- nepoužívať zápornú podmienku pri while ale radšej použiť until

Don't do this

```
while !@server.online?
```

```
end
```



```
# Useuntilstatement
```

```
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
```

```
forcolor in colors
```

```
putscolor
```

```
end
```

```
# Useeachblock
```

```
colors.each do |color|
```

```
putscolor
```

```
end
```

MVC

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

Kontroler

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

Model

- nepoužívajte zbytočné skratky pre názvy modelov (napríklad nie UserHstr ale UserHistory)
- neopakujte seba a ani nereplikujte funkcionality Rails/Ruby
- pokiaľ využívate rovnakú podmienku vo findviac krát, použite named_scope

```

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

# 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

# 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

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

# good
User.find(id)

  • taktiež preferujte find aj pri hľadaní podľa názvu atribútu

# bad
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`

```
# 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

```
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

`# Comment`

- umiestnite aspoň jednu medzeru medzi kódom a komentárom

`puts "Hi" # printsstring`

- dodržujte úroveň odsadenia, ktorú má kód

`defprintf(str)`

`# printsstring`

`puts "#{str}"`

`end`

- zarovnávajúce nasledujúce sa komentáre na konci riadku

`@variable # variablecomment`

`@longNameVariable # longnamevariablecomment`

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 zdefinujte v nasledovnom poradí
 - funkcionality
 - 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 zadaný vstup/výstup, tak umiestnite jeden prázdny komentárový riadok za opisom funkcionality metódy
- dátové typy vždy uvádzajte vo vnútri hranatých zátvoriek

Príklad

```
# ClassforallevilMonsters.

class Monster

  # Initwithnameofthe Monster.

  #

  # @paramname [String] Monster name.

  definitialize(name)

    @name = name.capitalize

  end

  # @return [String] Returningname.

  defgetName

    var_name = @name

    returnvar_name

  end

  # Revealtrue Monster identity.

  defreveal
```

```
puts "TheCallof #{getName}!"  
end  
end
```

YARD

- inštalácia

```
> gem install yard
```

- generuj dokumentáciu iba pre zdrojové súbory, kde nastala zmena

```
> yard docmonsterClass.rb
```

Metodika pre testovanie backendu pomocou RSpecu

Prečo testovať?

- menej bugov
- rýchlejšie a jednoduchšie manuálne pretestovanie
- jednoduchšie doimplementovanie novej funkcionality

Čo testovať v modeloch?

- validnúfactory

```
it "has a validfactory" 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 ".visiblereturnallvisibleprojects" 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

  • vztáhy 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 "publicinstancemethods" do
  context "responds to itsmethods" do
    it { expect(factory_instance).to respond_to(:public_method_name) }
  end

  context "executesmethodscorrectly" do
    context "#method_name" do
      it "doeswhatit'ssupposed to..." do
        expect(factory_instance.method_to_test).to eq(value_you_expect)
      end
    end
  end
end
end

```

```

describe "publicclassmethods" do
  context "responds to itsmethods" do
    it { expect(factory_instance).to respond_to(:public_method_name) }
  end
end

```

```

context "executes methods correctly" do
  context "self.methodname" do
    it "does what it's supposed to..." do
      expect(factory_instance.method_to_test).to eq(value_you_expect)
    end
  end
end
end
end

```

Čo testovať v kontroleroch?

- response

```

it "returns JSON of the project just updated" do
  expect(json[:name]).to eq @project_attributes[:name]
  expect(json[:description]).to eq @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 unittestouvrámcidescribe/context bloku, v ktorej je definovaný

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 /itwasborn in Belville/ }

GOOD -> možnosť využitia syntaxe is_expected.to

subject { assigns('message') }

it { is_expected.to match /itwasborn in Billville/ }

Describe, context, it

RSpec.describe "something" do

context "in onecontext" do

it "doesomething" do

end

end

context "in anothercontext" do

it "doesanotherthing" do

end

end

end

- pre describe použiť krátke a výstižné popisy metódy

```
describe '.class_method_name' do
```

```
  # codeblock
```

```
end
```

```
describe '#instance_method_name' do
```

```
  # codeblock
```

```
end
```

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

FactoryGirl + FFaker

```
FactoryGirl.define do
```

```
  factory :project do
```

```
    nameFFaker::Lorem.word
```

```
    descriptionFFaker::Lorem.sentence
```

```
    factory :deleted_project, class: Project do
```

```
      deletedtrue
```

```
    end
```

```
  end
```

```
end
```

- použitie takejto factory:

```
FactoryGirl.create :project
```

```
FactoryGirl.create :deleted_project
```

Užitočné zdroje

- kostra testov pre model (<https://gist.github.com/kylecarlson/6234923>)
- konvencie novej syntaxe v RSpec(<http://betterspecs.org/>)
- before/after bloky (<https://www.relishapp.com/rspec/rspec-core/v/3-0/docs/hooks/before-and-after-hooks>)
- ffakercheatsheet(<http://ricostacruz.com/cheatsheets/ffaker.html>)

- tutoriál na testovanie API (http://apiorails.icalialabs.com/book/chapter_three)
- tutoriál na factory(<https://robots.thoughtbot.com/aint-no-calla-back-girl>)

Metodika pre testovanie frontendu pomocou Jasmine

Unit testy

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

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('userloginform', 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 helpers shown for empty fields', function() {});
    it('ensure hitting enter on password field submits form', function() {
```

Konkrétny príklad testu:

```
describe("UnitTestingExamples", 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.toEqual(null);

// test cases - testing for success

var validEmails = [
    'test@test.com',
    'test@test.co.uk',
    'test734ltylytkliyt kryety9ef@jb-fe.com'
];

// test cases - testing for failure

var invalidEmails = [ 'test@test.com', '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 použi názov featury, ktorá sa ide testovať a v it krátko popíš funkcionality testu, začni slovesom.

Užitočné odkazy:

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

C Export evidencie úloh

Issue Type	Key	Summary	Assignee	Status
Task	UXWEB-69	Create a records for meeting 7 and 8	Roman Roba	Open
Sub-task	UXWEB-68	UXWEB-65 High fidelity template	Peter Dubec	Open
Sub-task	UXWEB-67	UXWEB-65 Low fidelity template	Peter Dubec	Resolved
Story	UXWEB-66	Flash messages	Monika Sanyova	Open
Story	UXWEB-65	Low/high fidelity web app template	Peter Dubec	Open
Story	UXWEB-64	Dashboard template	Tomas Zigo	In Progress
Story	UXWEB-63	Login template	Roman Roba	In Progress
Story	UXWEB-62	Work documentation	Robert Cuprik	Open
Story	UXWEB-61	Management Documentation	Jakub Benjamin Vrba	Open
Task	UXWEB-60	Create a records for meeting 5 and 6	Roman Roba	Closed
Story	UXWEB-59	Jira methodics	Roman Roba	In Progress
Sub-task	UXWEB-58	UXWEB-41 Grunt-Rails configuration	Patrik Gajdosik	Closed
Task	UXWEB-57	Low fidelity Project overview	<i>Unassigned</i>	Open
Story	UXWEB-56	RoR methodics	Peter Dubec	Closed
Sub-task	UXWEB-55	UXWEB-1 Routes	Robert Cuprik	Closed
Sub-task	UXWEB-54	UXWEB-1 Frontend tests, refactoring	Robert Cuprik	Closed
Sub-task	UXWEB-53	UXWEB-1 Tests for controller	Monika Sanyova	Closed
Sub-task	UXWEB-52	UXWEB-1 Change scope	Peter Dubec	Closed
Story	UXWEB-51	Register for TP CUP	Peter Dubec	Closed
Story	UXWEB-50	Angular documentation methodics	Jakub Benjamin Vrba	Open
Story	UXWEB-49	Code review methodics	Robert Cuprik	In Progress
Story	UXWEB-48	Deploy methodics	Patrik Gajdosik	Resolved
Story	UXWEB-47	RoR test methodics	Monika Sanyova	Closed
Story	UXWEB-46	JS tests methodics	Jakub Benjamin Vrba	Open
Story	UXWEB-45	RoR comments methodics + docs	Roman Roba	Closed
Story	UXWEB-44	JS comments methodics	Jakub Benjamin Vrba	Closed
Sub-task	UXWEB-43	UXWEB-41 SUBURI modification	Robert Cuprik	Closed
Sub-task	UXWEB-42	UXWEB-41 GRUNT automation	Jakub Benjamin Vrba	Closed
Story	UXWEB-41	GRUNT automation and SUBURI modification	Jakub Benjamin Vrba	Closed
Task	UXWEB-39	Remove libraries from repository	Patrik Gajdosik	Closed
Story	UXWEB-38	Flash messages	<i>Unassigned</i>	Open
Story	UXWEB-37	Mouse events	<i>Unassigned</i>	Open
Story	UXWEB-36	Location of target	<i>Unassigned</i>	Open
Story	UXWEB-35	Test display	<i>Unassigned</i>	Open
Story	UXWEB-34	Basic test CRUD	<i>Unassigned</i>	Open
Story	UXWEB-33	Tests list	<i>Unassigned</i>	Open
Story	UXWEB-32	Test start	<i>Unassigned</i>	Open
Story	UXWEB-31	Mouse coordinates	<i>Unassigned</i>	Open

Story	UXWEB-30	Target selection	<i>Unassigned</i>	Open
Story	UXWEB-29	New test	<i>Unassigned</i>	Open
Story	UXWEB-28	Image upload	Patrik Gajdosik	In Progress
Epic	UXWEB-27	We need to setup a project (repo, code, server, etc.)	<i>Unassigned</i>	Open
Epic	UXWEB-26	Customer wants to get basic statistics on the test	<i>Unassigned</i>	Open
Epic	UXWEB-25	Participant wants to solve a test	<i>Unassigned</i>	Open
Epic	UXWEB-24	Customer wants to search over projects, tests and users,	<i>Unassigned</i>	Open
Epic	UXWEB-23	Customer wants to manage his projects and tests	<i>Unassigned</i>	Open
Epic	UXWEB-22	Customer wants to create a test and assign it to participants	<i>Unassigned</i>	Open
Epic	UXWEB-21	User wants to register and log in using variety of possibilities (LDAP, password, Twitter, Google, ...)	<i>Unassigned</i>	Open
Task	UXWEB-20	Create a records for meeting 1, 2, 3 and 4	Roman Roba	Closed
Sub-task	UXWEB-19	UXWEB-12 Setup server for running the ux testing application	Patrik Gajdosik	Closed
Sub-task	UXWEB-18	UXWEB-12 Setup CI on the GitHub repository	Patrik Gajdosik	Closed
Sub-task	UXWEB-17	UXWEB-1 FRONTEND views for managing project	Tomas Zigo	Closed
Sub-task	UXWEB-16	UXWEB-12 Automate team web deployment	Patrik Gajdosik	Closed
Sub-task	UXWEB-15	UXWEB-12 Host our team web	Patrik Gajdosik	Closed
Sub-task	UXWEB-14	UXWEB-3 Low fidelity prototype	<i>Unassigned</i>	Closed
Sub-task	UXWEB-13	UXWEB-12 Install and Configure	Patrik Gajdosik	Closed
Task	UXWEB-12	Set up server	Patrik Gajdosik	Closed
Task	UXWEB-11	Create a basic structure for Angular	Jakub Benjamin Vrba	Closed
Task	UXWEB-10	Create basic structure for Rails API	Robert Cuprik	Closed
Task	UXWEB-9	Make a web page	Peter Dubec	Closed
Task	UXWEB-8	Create a Git repo	Peter Dubec	Closed
Task	UXWEB-7	Git methodics	Monika Sanyova	Closed
Sub-task	UXWEB-6	UXWEB-1 Backend API for index, create, update, destroy	Monika Sanyova	Closed
Story	UXWEB-4	UXWEB-2: LDAP login	Robert Cuprik	In Progress
Story	UXWEB-3	High fidelity dashboard prototyp	Tomas Zigo	Closed
Story	UXWEB-2	Allow user to log in	Robert Cuprik	Closed
Story	UXWEB-1	Basic project management(CRUD)	Monika Sanyova	Closed