Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Editor

Tím 24

Študijný program: Inteligentné softvérové systémy, Internetové technológie
Ročník: 1
Predmet: Tímový projekt
Cvičiaci: Ing. Eduard Kuric, PhD.

Obsah

1 Úvod ... 4

2 Analýza ... 5

2.1 Analýza podporného nástroja pre tvorbu stránok ... 6

2.1.1 Diagram tried, CSS composer ... 6

2.1.2 Sekvenčný diagram, zmena textového komponentu ... 7

2.1.3 Diagram tried, manažér selektorov ... 8

2.1.4 Sekvenčný diagram, pridanie komponentu do kolekcie komponentov ... 9

3 Návrh ... 10

3.1 Rozšírenie návrhu editora ... 14

4 Implementácia ... 18

4.1 Editor bez kolaborácie ... 18

4.1.1 Akceptačné kritéria ... 19

4.1.2 Validácia ... 19

4.1.3 Dátový model ... 20

4.1.4 Autorizácia ... 21

4.2 Kolaboratívny editor ... 22

4.2.1 Akceptačné kritéria ... 22

4.2.2 Validácia ... 23

4.2.3 Práca s prototypmi ... 24

4.2.4 Synchronizácia zmien v prototypoch ... 28

4.2.5 Vizualizácia označených komponentov ... 37

4.2.6 Uzamykanie komponentov ... 43

5 Testovanie ... 48

Obsah obrázkov

Obrázok 1 - Diagram tried pre CSS Composer 6
Obrázok 2 - Sekvenčný diagram pre zmenu textového komponentu 7
Obrázok 3 - Diagram tried pre manažér selektorov 8
Obrázok 4 - Diagram tried pre pridanie komponentu do kolekcie komponentov 9
Obrázok 5 - UC diagram pre editor 10
Obrázok 6 - UC diagram pre rozšírenie návrhu editora 14
Obrázok 7 - Databázový Model pre editor 20
Obrázok 8 - NoSQL databázový model pre editor 20
Obrázok 9 - Obrazovka pre vytvorenie novej stránky 24
Obrázok 10 - Obrazovka pre mazanie stránky 25
Obrázok 11 - Obrazovka pre panel kolaborantov 26
Obrázok 12 - Obrazovka pre panel stránok 27
Obrázok 13 - Posielanie zmien vykonaných v editore ostatným klientom (všeobecný prípad) 28
Obrázok 14 - Udalosti súvisiace s označením komponentu v editore po jeho načítaní 37
Obrázok 15 - Zvýraznenie komponentu, ktorý je práve označený iným kolaborantom 38
Obrázok 16 - Zvýraznenie označených komponentov medzi dvoma rôznymi kolaborantami súčasne 38
Obrázok 17 - Uzamykanie/odomykanie komponentov 43
Obrázok 18 - Odomknutý komponent 44
Obrázok 19 - Uzamknutý komponent 44
Obrázok 20 - Zoznam uzamknutých komponentov 44

1 Úvod

Ďalší bod, ktorým sme sa zaoberali v rámci projektu CollabUI je samotný editor. Rozdelili sme ho na 2 časti a tými sú:

- Editor bez kolaborácie
- Kolaboratívny editor

V rámci editora oslobodeného od kolaborácie bolo nutné vyriešiť niekoľko problémových oblastí. Jednou s nich je konfigurácia podporného nástroja pre tvorbu stránok, ktorý predstavuje jadro pre úpravu a vytváranie jednotlivých stránok. Pomocou neho bude môcť používateľ lepšie štýlovať stránky bez nutnosti ovládania programátorských zručností. Elementy na stránke sú reprezentované ako komponenty a práve preto sme museli vyriešiť aj ukladanie týchto komponentov aby sme neskôr vedeli uchovávať prototyp, na ktorom vlastník projektu alebo kolaborant pracoval. Pri príchode na projekt sa používateľovi zobrazí základný prototyp rozloženia stránky, ktorý slúži na iniciálne oboznámenie sa s nástrojom. Tu používateľ môže vidieť akým štýlom môžu byť rôzne komponenty kombinované a taktiež ako ich meniť. Ďalšou obšírnou časťou editora tvoria bočné panely nástroja. Používateľ bude mať možnosť si v nastaveniach vybrať, ktoré panely sa budú nachádzať v pravej alebo ľavej časti editora. Taktiež si môže prispôsobovať ich veľkosť podľa vlastných potrieb. Ďalej sme implementovali obmedzenia pre role „watch“ a role „banned“.

Čo sa týka editora s implementáciou kolaborácie sme dorábali jednotlivé časti akými bolo preposielanie dát kolaborantov, synchronizácia zmien medzi kolaborantami a mnoho iného. Detailne nižšie.

2 Analýza

V rámci analýzy sme riešili najmä funkcionality knižnice pre tvorbu stránok. Zistili sme, že keďže je táto knižnica ešte relatívne nová, tak neponúka veľa funkcii čo sa týka ukladaní a prácou s komponentami. Pri ukladaní sa používa vstavaný objekt storageManager. Tento objekt ponúka funkcie akými sú auto ukladanie, získavanie predchádzajúcich krokov pri zmene komponentov, získavanie samotného komponentu ako aj jeho pridávanie. Taktiež sme zistili, že tento objekt pri ukladaní komponentov vždy pracuje so všetkými komponentami, ktoré sa v editore nachádzajú. Táto skutočnosť nie je pre nás najvýhodnejšia, keďže potrebujeme narábať často krát len s jedným konkrétnym komponentom. Riešenie by mohlo spočívať v pre-iterovaní všetkých komponentov pri posielaní do editora a následnou selekciou tých špecifických komponentov, s ktorými používateľ potrebuje pracovať.

Taktiež sme identifikovali skutočnosť, že na mobilných zariadeniach nebude možné efektívne pracovať s editorom kvôli nízkemu rozlíšeniu a komplexnosti riešenia, práve z tohoto dôvodu sme zavrhli podporu mobilných zariadení v rámci editora a vytvorili novú chybovú stránku.

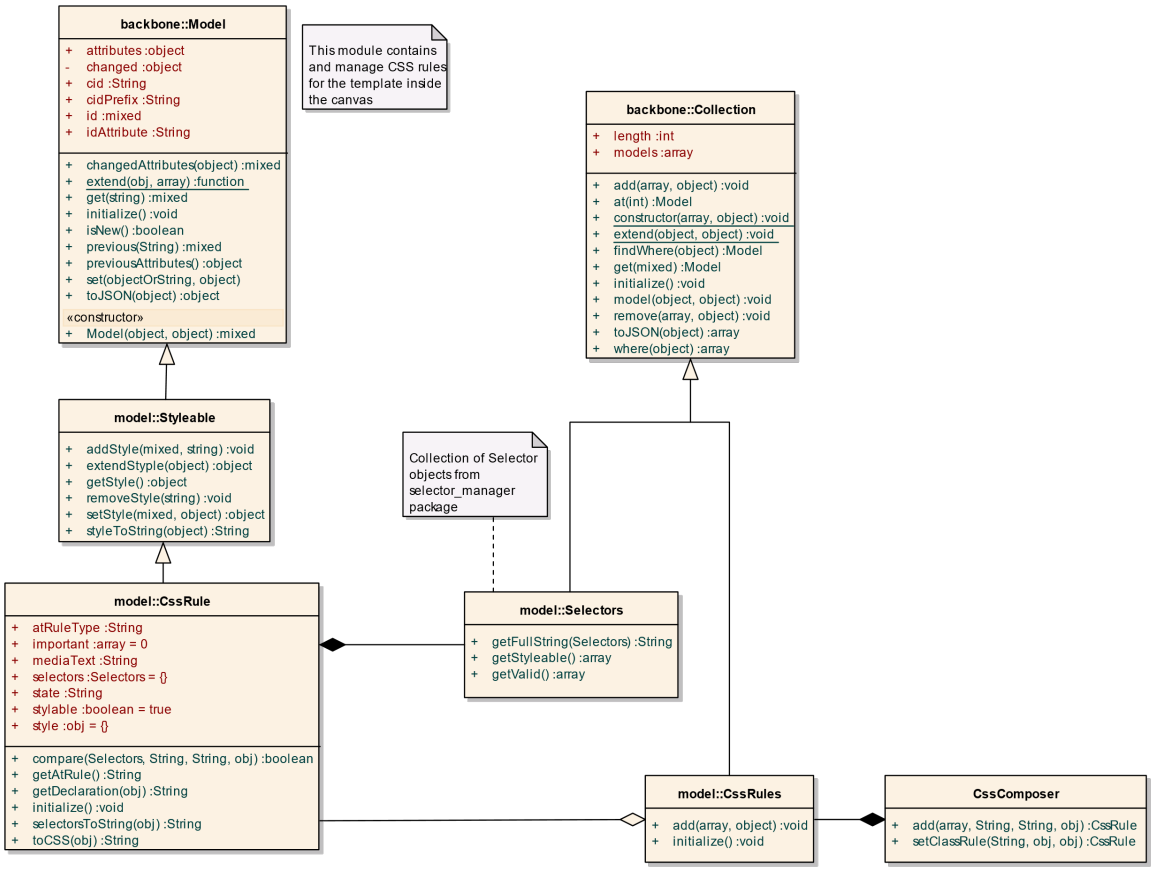
Analýza riešenia kolaborácie v reálnom čase vyžadovala hlbšie ponorenie sa do technickej štruktúry podporného nástroja pre tvorbu stránok, ktorej opis sa nachádza v nasledujúcich častiach.

2.1 Analýza podporného nástroja pre tvorbu stránok

K pochopeniu vnútorného fungovania podporného nástroja pre tvorbu stránok[1] bolo nutné zhotoviť UML diagramy. Znalosti, ktoré sme z diagramov nadobudli využívame pri implementácií kolaboratívnej časti projektu, nakoľko samotná kolaborácia si vyžaduje komplexnejšie úpravy GrapesJS a preto pochopenie vnútorného fungovania niektorých častí bolo nevyhnutnou podmienkou pre ich správnu implementáciu. Sústredili sme sa na označovanie komponentov, CSS tried komponentov, zmeny textového komponentu a pridanie nového komponentu.

2.1.1 Diagram tried, CSS composer

Diagram tried CSS ovládača, ktorý zachytáva vzťahy medzi triedami Backbone.js a GrapesJS. BackboneJS je framework určený na štruktúrovanie JavaScript kódu do MVC modelu, ktorý GrapesJS využíva na reprezentáciu.

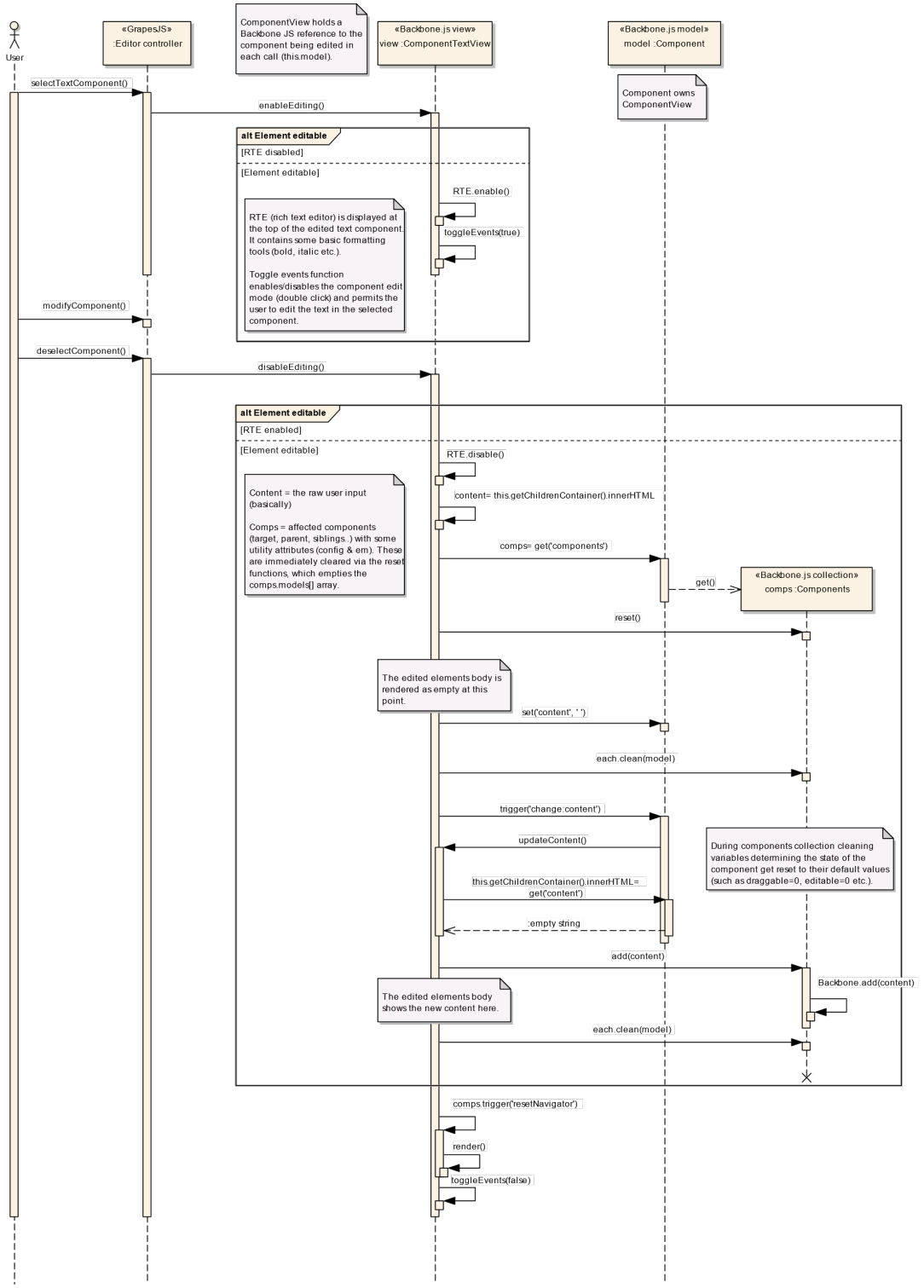


Obrázok 1 - Diagram tried pre CSS Composer

[1] http://grapesjs.com/

2.1.2 Sekvenčný diagram, zmena textového komponentu

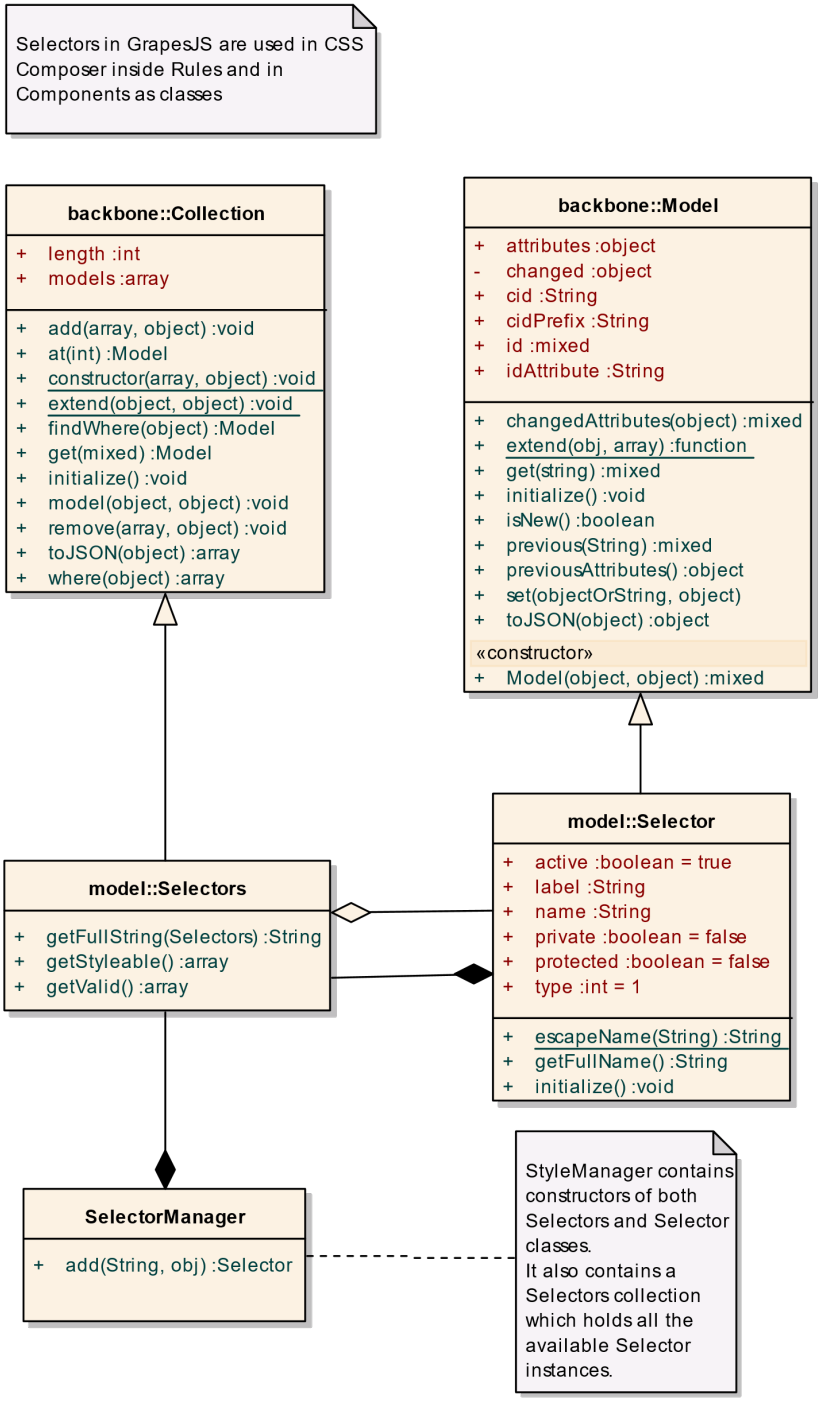
Sekvenčný diagram, ktorý zachytáva tok udalostí pri zmene textového komponentu.



Obrázok 2 - Sekvenčný diagram pre zmenu textového komponentu

2.1.3 Diagram tried, manažér selektorov

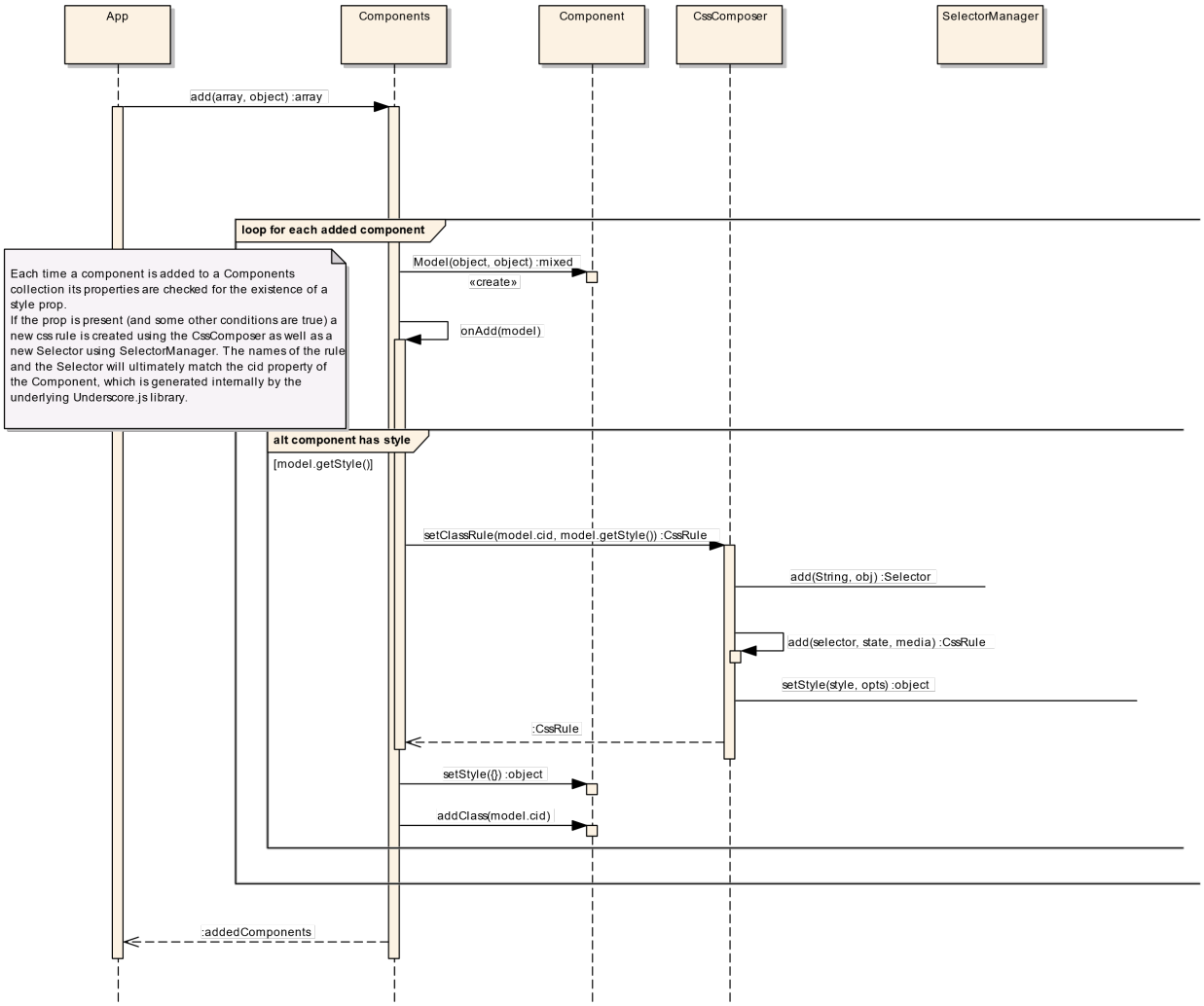
Diagram tried selektorov komponentov, ktorý zachytáva vzťah Backbone.js a GrapesJS. Rovnako ako pri CSS composer aj tu GrapesJs využíva Backbone.js na prácu s modelmi.



Obrázok 3 - Diagram tried pre manažér selektorov

2.1.4 Sekvenčný diagram, pridanie komponentu do kolekcie komponentov

Sekvenčný diagram, ktorý zachytáva tok udalostí pri pridaní komponentu do kolekcie komponentov.



Obrázok 4 - Diagram tried pre pridanie komponentu do kolekcie komponentov

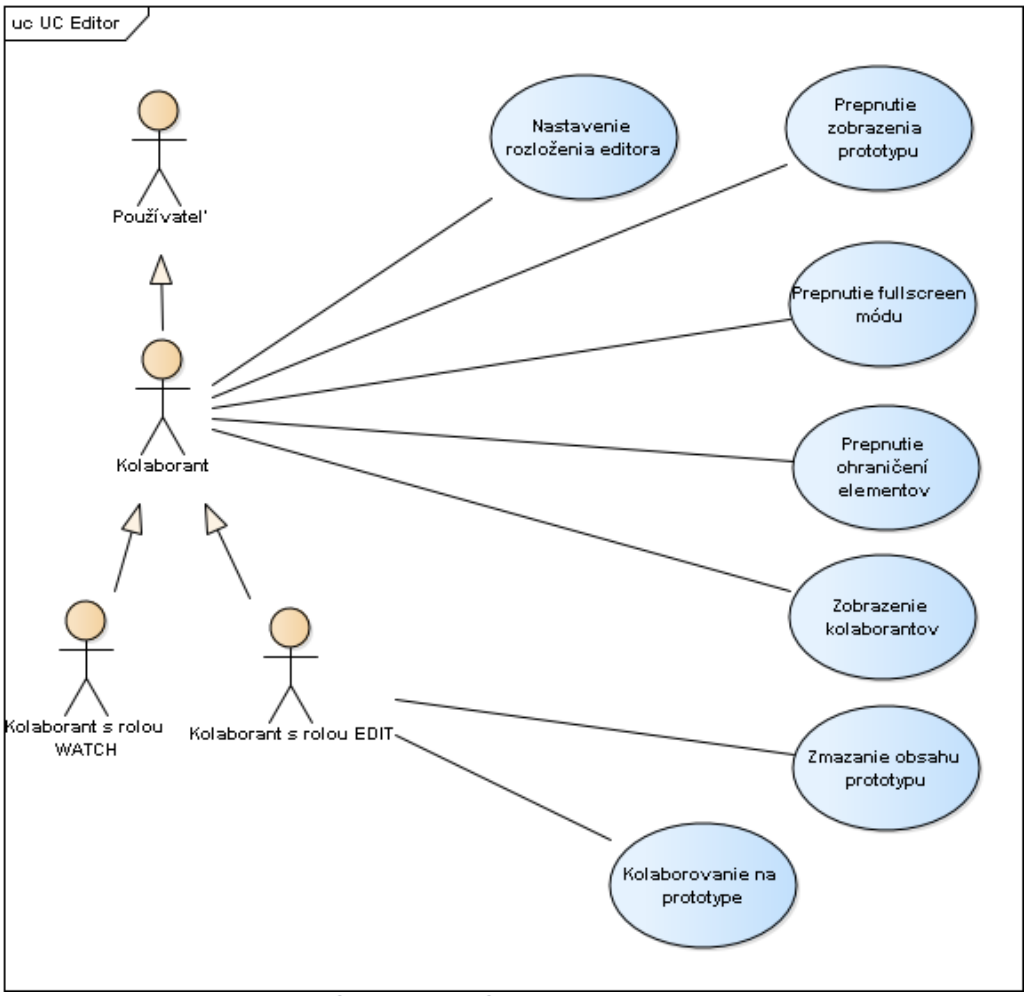
Diagramy, ktoré sme analyzovali nám v niektorých častiach pomohli k úspešnej implementácií. Treba však zdôrazniť, že samotné zakreslenie a analyzovanie diagramov je časovo veľmi náročné a preto je najlepšie zachytiť diagramom len tie najproblematickejšie časti. V rámci projektu sme chceli zakresliť ešte viacero diagramov, ktoré by nám v konečnom dôsledku mohli zrýchliť čas vývoja alebo pomôcť optimalizovať niektoré existujúce časti avšak vyššiu prioritu sme priradili kompletizácií celkovej implementácie projektu. Diagramy, ktoré odporúčame zakresliť a analyzovať do budúcna sú diagramy týkajúce sa zmien pozícií komponentov.

3 Návrh

Samotný editor sa skladá z nasledujúcich UC:

- Nastavenie rozloženia editora
- Prepnutie zobrazenia prototypu
- Prepnutie fullscreen módu
- Prepnutie ohraničení elementov
- Zmazanie obsahu prototypu
- Zobrazenie kolaborantov
- Kolaborovanie na prototype

Nasledujúci use case diagram znázorňuje akcie používateľa pri práci s editorom.



Obrázok 5 - UC diagram pre editor

UC Nastavenie rozloženia editora

Hlavný tok:

1. Používateľ zvolí možnosť “Prispôsobenie nastavení”
2. Používateľ nastaví umiestnenie jednotlivých nástrojov
3. Používateľ zvolí možnosť “Aktualizovať”
4. Systém zreviduje zmeny a podľa zvolených umiestnení nástrojov ich aktualizuje v editore
5. Systém notifikuje používateľa o vykonaných zmenách.
6. Prípad použitia končí.

UC Prepnutie zobrazenia prototypu

Hlavný tok:

1. Používateľ zvolí možnosť “Prepnutie zobrazenia” z 3 možností:
    - Mobil
    - Tablet
    - Desktop
2. Systém prepne zobrazenie prototypu podľa zvolenej možnosti
3. Prípad použitia končí

UC Prepnutie fullscreen módu

Hlavný tok:

1. Používateľ zvolí možnosť “Prepnutie fullscreen”
2. Systém prepne prototyp do režimu celého okna
3. Prípad použitia končí

UC Prepnutie ohraničení elementov

Hlavný tok:

1. Používateľ zvolí možnosť “Ohraničenie elementov”
2. Systém zobrazí ohraničenia elementov
3. Prípad použitia končí

UC Zmazanie obsahu prototypu

Hlavný tok:

1. Používateľ zvolí možnosť “Vyčistiť obsah prototypu”
2. Systém overí akciu používateľa prostredníctvom modálneho okna
3. Používateľ potvrdí akciu
4. Systém vymaže komponenty nastavené pre konkrétny projekt a uloží aktuálny stav
5. Prípad použitia končí

UC Zobrazenie kolaborantov

Hlavný tok:

1. Systém pri inicializácii editora prihlási používateľa
2. Systém notifikuje ostatných prihlásených používateľov o príchode nového kolaboranta
3. Systém zobrazí kolaboranta v panely určenom pre evidenciu kolaborantov
4. Prípad použitia končí

UC Kolaborovanie na prototype

Hlavný tok:

1. Systém pri inicializácii editora prihlási používateľa

2. Používateľ zvolí element na prototype alebo potiahne element z panelu preddefinovaných blokov do prototypu

3. Systém uloží vykonanú zmenu na prototype

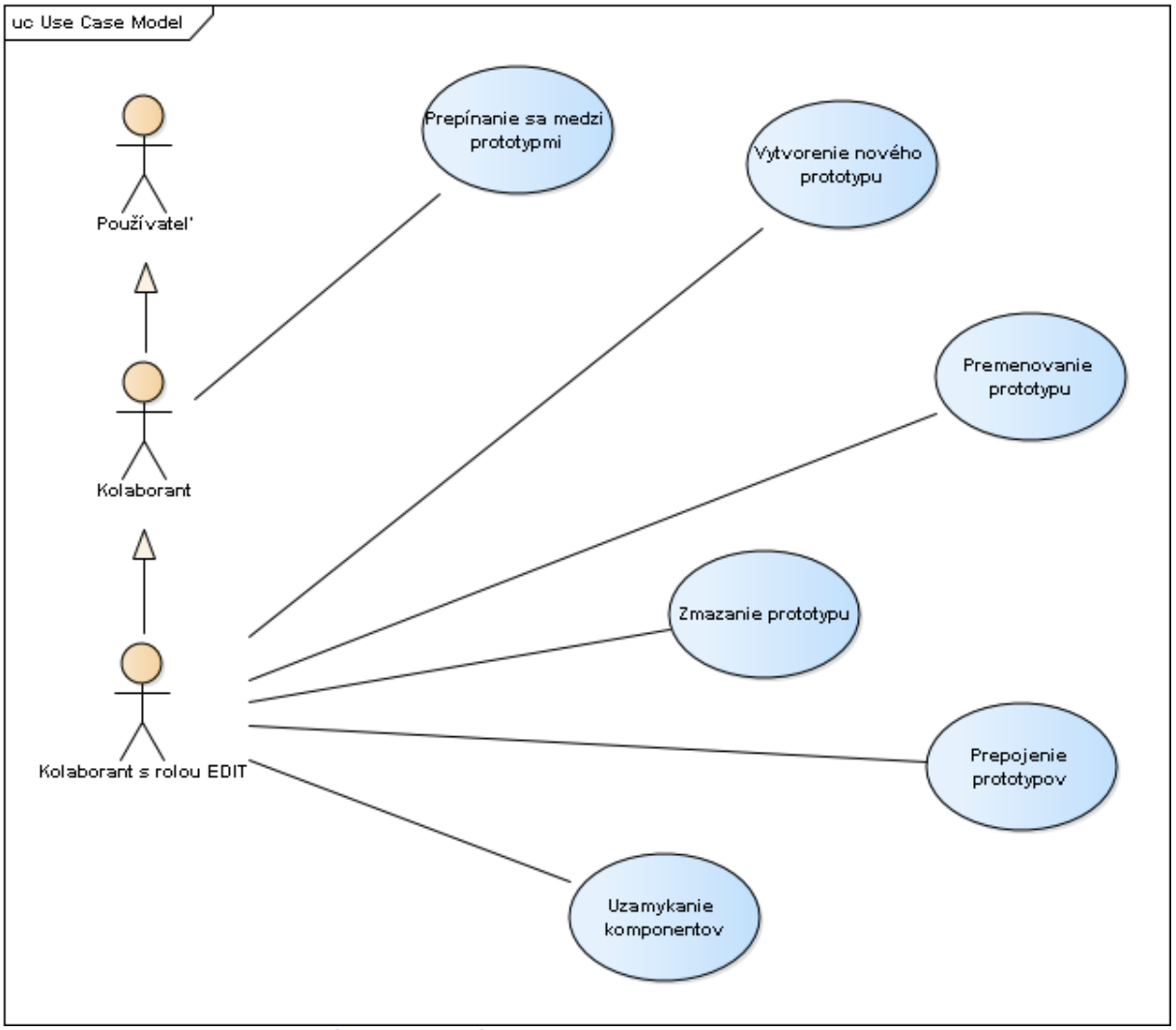
4. Systém odošle vykonanú zmenu všetkým prihláseným kolaborantom a aktualizuje ich lokálny prototyp v reálnom čase

5. Prípad použitia končí

3.1 Rozšírenie návrhu editora

Samotné rozšírenie návrhu editora sa skladá z nasledujúcich UC:

- Vytvorenie nového prototypu
- Premenovanie prototypu
- Zmazanie prototypu
- Prepínanie sa medzi prototypmi
- Prepojenie prototypov
- Uzamykanie komponentov



Obrázok 6 - UC diagram pre rozšírenie návrhu editora

UC Vytvorenie nového prototypu

Hlavný tok:

1. Kolaborant vyberie možnosť “New” v editore
2. Systém vytvorí nový prototyp, ďalej stránku a prepne kolaboranta na novo vytvorenú stránku
3. Systém notifikuje ostatných kolaborantov aby sa im vytvorila rovnaká stránka
4. Prípad použitia končí

UC Zmazanie prototypu

Hlavný tok:

1. Kolaborant vyberie možnosť odpadkového koša v paneli pre stránky
2. Systém zobrazí modálne okno s potvrdením o zmazanie
3. Kolaborant potvrdí akciu zmazania stránky
4. Systém vymaže stránku a notifikuje ostatných kolaborantov aby sa im stránka rovnako zmazala
5. Prípad použitia končí

Alternatívny tok:

3.a1 Kolaborant zruší akciu vymazania stránky

3.a2 Systém nevykoná žiadnu akciu

3.a3 Pokračuje bodom 5

UC Premenovanie prototypu

Hlavný tok:

1. Kolaborant vyberie možnosť ceruzky v paneli pre stránky
2. Systém zobrazí modálne okno s potvrdením o premenovanie
3. Kolaborant potvrdí akciu premenovania stránky
4. Systém premenuje stránku a notifikuje ostatných kolaborantov aby sa im stránka rovnako premenovala
5. Prípad použitia končí

Alternatívny tok:

3.a1 Kolaborant zruší akciu premenovania stránky

3.a2 Systém nevykoná žiadnu akciu

3.a3 Pokračuje bodom 5

UC Prepínanie medzi prototypmi

Hlavný tok:

1. Kolaborant zvolí jednu z existujúcich stránok v editore
2. Systém získa informácie o stránke
3. Systém spracuje a zobrazí stránku v korektnej podobe
4. Systém notifikuje ostatných kolaborantov o prepnutí kolaboranta na konkrétnu stránku
5. Prípad použitia končí

UC Prepojanie prototypov

Hlavný tok:

1. Kolaborant vyberie komponent “anchor” z panelu preddefinovaných komponentov a potiahne ho do editora

2. Kolaborant vyberie v panely konfigurácie komponentu jeden z vytvorených prototypov, ktorý chce s otvoreným prototypom prepojiť

3. Systém ch pripojených notifikuje ostatných pripojených kolaborantov o zmene

4. Prípad použitia končí

UC Uzamykanie komponentov

Hlavný tok:

1. Kolaborant vyberie možnosť "uzamknutie komponentu“ (zámok) na danom komponente

2. Systém uzatvorí možnosť editácie komponentu pre ostatných kolaborantov a notifikuje ich o zmene

3. Systém indikuje uzamknuté komponenty v panely vrstiev

4. Prípad použitia končí

4 Implementácia

4.1 Editor bez kolaborácie

Základným stavebným kameňom celého projektu je editor. V samotnom editore bude môcť používateľ vykonávať zmeny v rámci konkrétneho prototypu. V súčasnej implementácii neuvažujeme o simultánnom vytváraní viacerých prototypov ani o verziovaní. Základ editora tvorí podporná knižnica pre vytváranie prototypov vo webovom rozhraní, ktorej funkcionality využívame. Knižnica disponuje 4 vstavanými panelmi:

- Panel pre pridanie preddefinovaných elementov
- Panel pre úpravu štýlov
- Panel vrstiev
- Panel konfigurácie komponentov

Naraz však dokáže knižnica pracovať iba s jedným panelom. Túto skutočnosť sme vyriešili vlastnou implementáciou, ktorá dovoľuje pridať ďalšie panely. Kvôli budúceho rozširovania editora sme pred pripravili nasledujúce panely:

- Panel pre zoznam prototypov (stránok)
- Panel pre zoznam poznámok
- Panel pre históriu akcii
- Panel pre zoznam kolaborantov
- Panel pre chat
- Panel pre uzamknuté komponenty

Taktiež sme mysleli na potrebu používateľa prispôsobiť rozloženie editora a z tohoto dôvodu sme pridali možnosť nastavenia umiestnenia jednotlivých panelov a to:

- Vpravo hore
- Vpravo dole
- Vľavo hore
- Vľavo dole

Panely taktiež umožňujú zmeniť veľkosť a v prípade, že používateľ nevložil do niektorého umiestnenia ani jeden panel, dané umiestnenie sa skryje a editor sa prispôsobí k tejto zmene. Rozloženie editora je možné uložiť pre právo „WATCH“ a zvlášť pre „EDIT“, totiž pri práve „WATCH“ je používateľ obmedzený len na 4 panely, vstavané panely knižnice sa v tomto prípade nevyužívajú.

Obmedzenia pre kolaborantov s právom WATCH sú nasledovné:

- Zmazanie obsahu prototypu
- Úprava, editovanie prototypu
- 4 vstavané panely knižnice
- Vytvorenie, premenovanie, zmazanie prototypu
- Uzamykanie komponentov

Ďalším obmedzením je stav BANNED. Používateľ s týmto obmedzením je po prístupe na editor presmerovaný na chybovú stránku.

Ukladanie prototypu prebieha automaticky po každej zmene na strane NodeJS Servera, pričom dáta sa kvôli veľkosti ukladajú do rýchlej nerelačnej databázy. K ukladaniu prototypu sme využili JSON reprezentáciu, tak ako ju definuje podporná knižnica.

Funkcionality ako prepnutie zobrazenia prototypu, fullscreen, či ohraničenia elementov sme prepoužili z podpornej knižnice.

4.1.1 Akceptačné kritéria

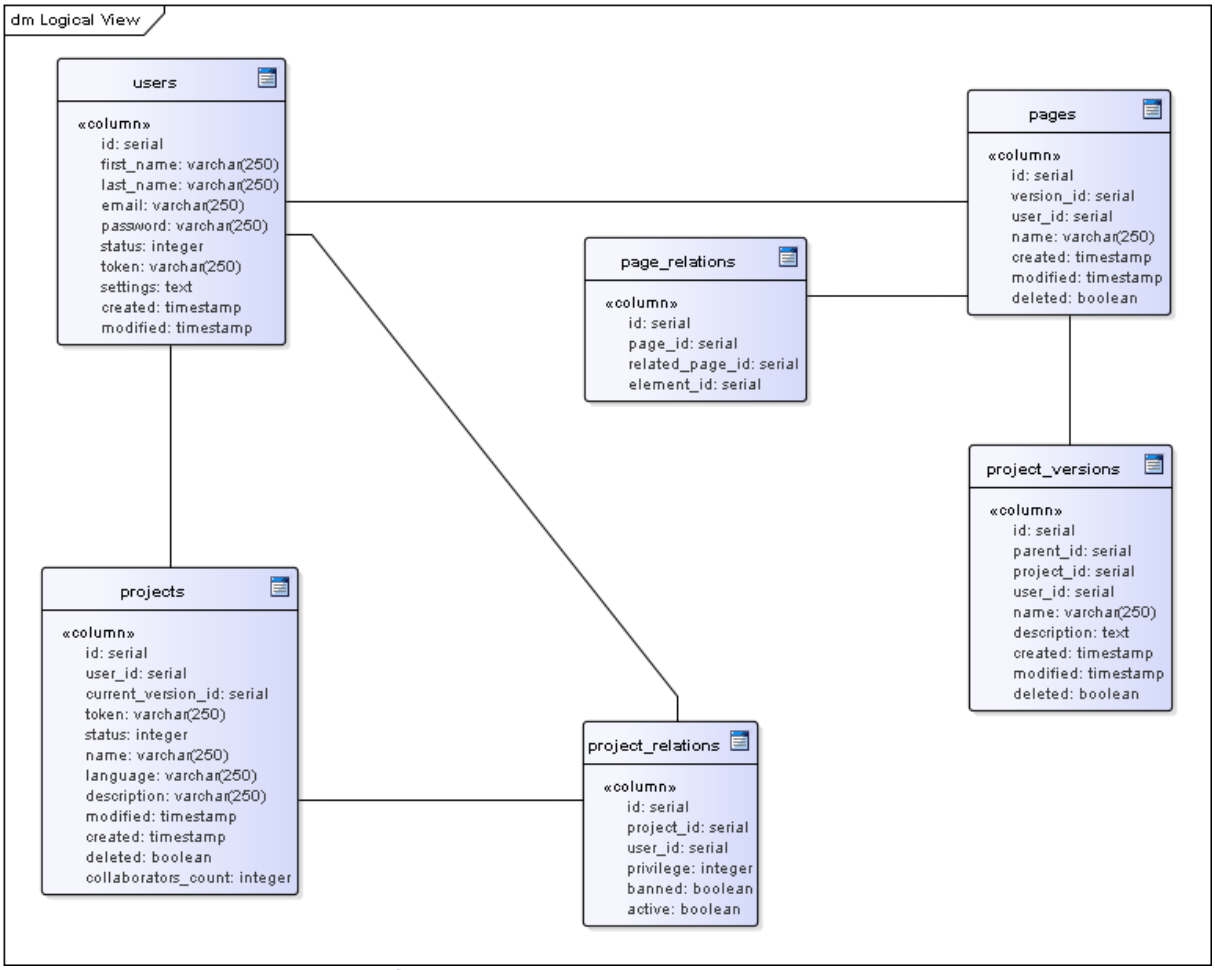
- Možnosť prihlásenia sa do editora a následná autorizácia
- Vyskladanie prototypu prostredníctvom editora
- Vytvorený prototyp je automaticky uložený po zmene
- Možnosť zmeny rozloženia panelov editora pre právo „EDIT“ a zvlášť pre právo „WATCH“
- Používateľ s právom WATCH, nedokáže editovať prototype alebo vykonať akúkoľvek zmenu, ktorú by systém zachytil a uložil
- Používateľ s právom BANNED je presmerovaný na chybovú stránku

4.1.2 Validácia

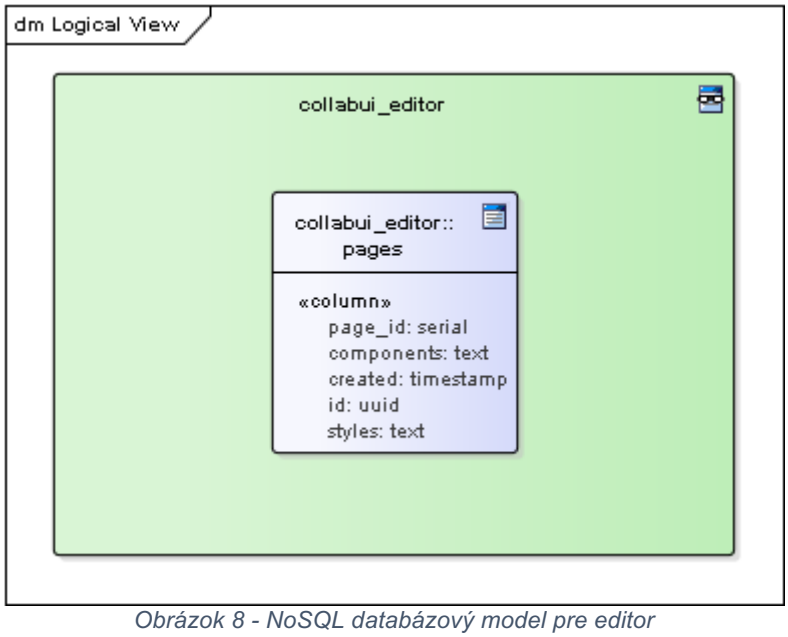
Pri editore je dôležité hlavne validovať skutočnosť, že používateľ s právom WATCH nesmie za žiadnych okolností upraviť prototyp. Validácia prebieha na strane NodeJS Servera.

4.1.3 Dátový model

Poznámka: Zelený štvorec v NoSQL diagrame znázorňuje keyspace.



Obrázok 7 - Databázový Model pre editor



Obrázok 8 - NoSQL databázový model pre editor

4.1.4 Autorizácia

Autorizácia v editore prebieha prostredníctvom vygenerovaných token, s ktorými disponuje každý používateľ a každý projekt.

Príklad URL: editor/@922b9fcf0ef4ab53/41/@4637f2ca7eed4f83

Tokeny začínajú znakom @.

- Token projektu
- ID verzie projektu
- Token používateľa

Pri prístupe na editor s danou URL sa identifikuje projekt a na základe uloženého vzťahu projekt-používateľ sa identifikujú používatelia, ktorí majú prístup. Potom na základe tokenu používateľa vyberie zo zoznamu povolených používateľov konkrétny s právom „EDIT“ alebo „WATCH“. Ak tokeny nekorešpondujú hore uvedenej skutočnosti, prístup sa vyhlási za neautorizovaný a používateľovi je zobrazená chybová stránka.

Ak prvou úrovňou autorizácie používateľ prešiel úspešne zobrazíme mu editor. Po inicializácii editora nadviaže systém komunikáciu s NodeJS Serverom kde prebieha druhá úroveň autorizácie rovnakým princípom.

4.2 Kolaboratívny editor

Po implementácii základnej kostry editora sme pristúpili k implementácii kolaboratívneho módu, čo je hlavným poslaním vyvíjaného systému. K implementácii zmien v prototype v reálnom čase sme využili NodeJS Server s kombináciou so socket.io knižnicou prostredníctvom, ktorej posielame upravené elementy na server, kde sa dáta rozposielajú cez broadcast ostatným pripojeným kolaborantom, hneď po uložení zmien v rýchlej nerelačnej databáze Cassandra. Taktiež zachytávame udalosť označenia elementu používateľom, ktorú rozposielame cez broadcast kolaborantom, pričom sa im daný element vizuálne ohraničí.

Po prihlásení alebo odhlásení používateľa z editora, taktiež notifikujeme ostatných kolaborantov formou notifikácie, zároveň aktualizujeme panel pre zoznam kolaborantov, kde vizuálne odlišujeme pripojených od nepripojených kolaborantov. Taktiež sme zapracovali podporu viacerých simultánne editovateľných prototypov a prácu s nimi. Pre každého kolaboranta udržujeme informáciu, na ktorom prototype aktuálne pracuje, pomocou čoho dokážeme kategorizovať pripojených kolaborantov.

Ďalej sme sa sústredili na vyladenie kolaborácie a synchronizáciu zmien v prototypoch. Počas testovania v rámci implementácie sme narazili na problém s vymazanými komponentami, ktoré boli označené iným kolaborantom, preto sme začali uvažovať nad možnosťou uzamykania komponentov o čom sa dočítate nižšie.

4.2.1 Akceptačné kritéria

- Používateľ dokáže vytvoriť novú stránku
    - Po vytvorení stránky je používateľ presmerovaný na novo vytvorenú stránku
    - Po vytvorení stránky musia byť všetci kolaboranti notifikovaní
- Používateľ dokáže meniť obsah novej stránky bez ovplyvnenia ostatných stránok
- Používateľ dokáže premenovať stránku
    - Premenovanie stránky musím najskôr potvrdiť
    - Po premenovaní stránky musia byť všetci kolaboranti notifikovaní
- Používateľ dokáže odstrániť stránku
    - Odstránenie stránky musím najskôr potvrdiť
    - Po odstránení bude používateľ presmerovaný na stránku na ľavo od vymazanej
    - Po vymazaní stránky sa stránka (a jej vzťah) reálne nezmažú iba sa označia v DB ako deleted
    - Po vymazaní stránky musia byť všetci kolaboranti notifikovaní
- Používateľ sa dokáže prepínať medzi stránkami
    - Pri prepnutí na inú stránku sa kolaborant zobrazí v paneli kolaborantov na danej stránke
    - Pri prepnutí na inú stránku sa zobrazia dáta konkrétnej stránky
- Používateľ dokáže kolaborovať s ostatnými kolaborantami
    - Používateľ vidí v reálnom čase vykonané zmeny na prototype inými kolaborantami

    - Používateľ vidí aktuálne označené elementy prototypu ostatných kolaborantov
    - Používateľ je notifikovaný pri príchode a odchode kolaboranta
    - Používateľ má prístup k zoznamu aktuálne prihlásených kolaborantov v editore
    -
- Používateľ dokáže prepojiť vytvorené stránky
- Používateľ dokáže uzamknúť vybraný komponent alebo skupinu komponentov

4.2.2 Validácia

Vytvorenie stránky sa vykoná iba v prípade, že prebehne úspešný dopyt na vytvorenie záznamu.

Premenovanie stránky sa vykoná iba v prípade, že prebehne úspešný dopyt na premenovanie záznamu.

Pri vymazaní stránky je potrebné overiť či kolaborant, ktorý sa snaží vymazať stránku má právo na túto akciu, konkrétne aby nevymazal stránku na inom projekte. Základnú index stránku nie je možné vymazať, čo je kontrolované pred dopytom do databázy.

4.2.3 Práca s prototypmi

Na vytváranie, mazanie, prepínanie stránok a notifikovanie o akciách sa používajú správy posielané pomocou NodeJS a pre ukladanie informácií o stránkach(komponenty a štýly) sa používa CQL(Cassandra) databáza.

Komponenty a štýly sú v CQL databáze uložené a sprístupnené cez ich ID a neexistuje vzťah medzi stránkou a projektom. Neoprávnené prístupy na modifikáciu stránky sú ošetrené na strane NodeJS servera, ktorý kontroluje na akej stránke sa nachádza kolaborant a teda či má právo modifikovať stránku.

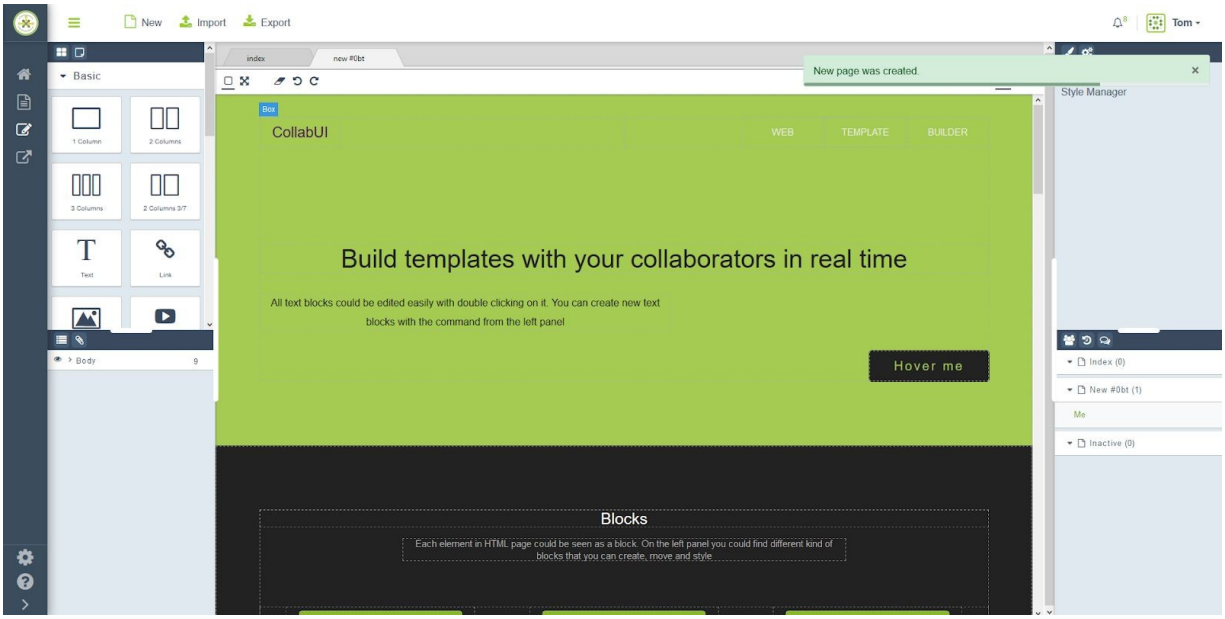
Vytvorenie stránky

Ako prihlásený používateľ v editore chcem vedieť vytvoriť novú stránku. Vytváranie novej stránky sa vykonáva v editore v konkrétnom projekte.

Zoznam vytvorených stránok sa nachádza v SQL databáze a pristupuje sa k nim cez id.

Pri kliknutí na vytvorenie novej stránky sa otvorí modálne okno, do ktorého je potrebné zadať názov novej stránky. Potvrdenie akcie v modálnom okne zavolá dopyt do databázy a po úspešnom zápise sa vytvorí nová karta vo vrchnej časti editora a notifikujú sa ostatní kolaboranti. Tento proces sa nachádza v *panelPrototypes.js* vo funkcii *createPage*.

Po tomto procese je vytvorená stránka načítaná aj do oboch bočných panelov, konkrétne do panelu Prototypes a Collaborators.

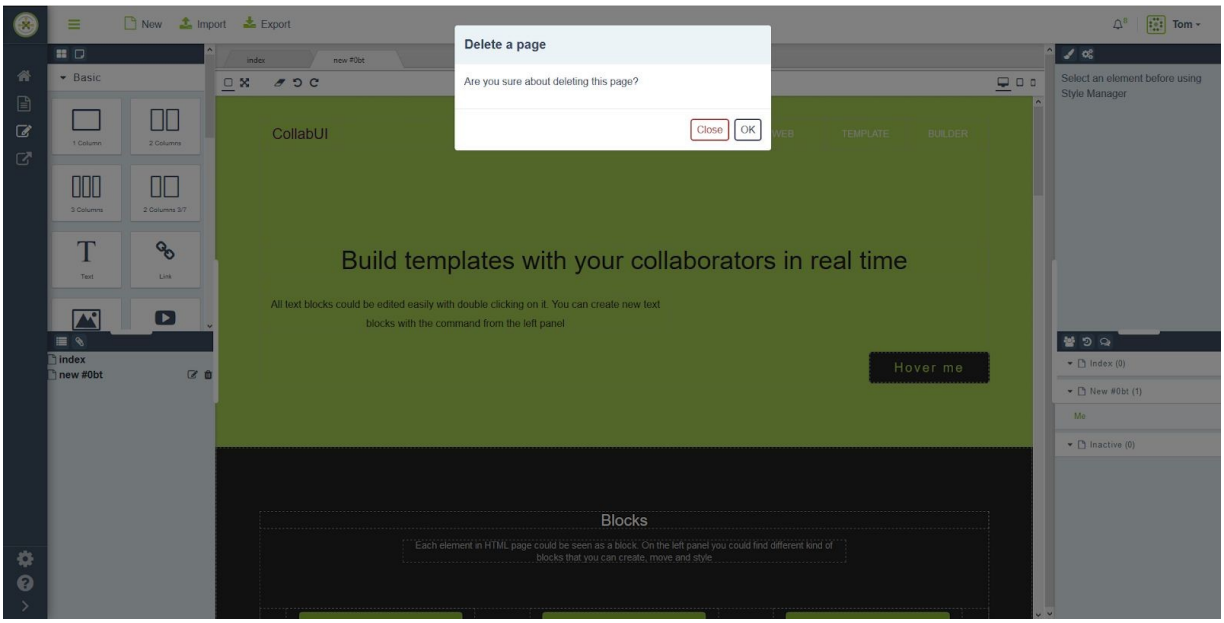


Obrázok 9 - Obrazovka pre vytvorenie novej stránky

Zmazanie stránky

V tomto prípade použitia sa odstráni stránka u kolaboranta, ktorý túto akciu inicioval a notifikujú sa ostatní kolaboranti aby sa aj im stránka odstránila z editora.

V rámci budúcej práce by sme chceli zaintegrovať databázový cron, ktorý by každý deň o polnoci prešiel všetky záznamy, záznamy s flagom deleted by zarchivoval a v poslednom kroku vymazal z databázy.



Obrázok 10 - Obrazovka pre mazanie stránky

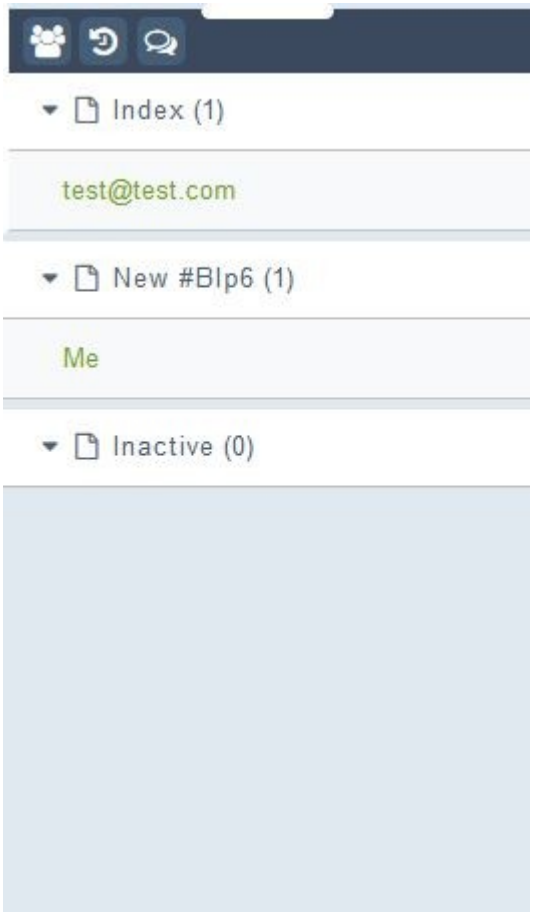
Prepínanie medzi stránkami

Prepínanie medzi stránkami používa NodeJS server na notifikovanie a načítavanie obsahu inej stránky. Kolaboranti sú zobrazovaní na akej stránke sú momentálne aktívny, táto informácia sa nachádza v paneli kolaborantov.

Pri prepnutí na inú stránku sa z databázy získajú dáta o stránke(komponenty a štýly) a načítajú sa do editora v *EditorClient.js.* Počas načítavania sa pozastavia všetky event listenery(napr. component:add) a po načítaní sa obnovia. Pozastavenie sa vykonáva pomocou nastavenia boolean premennej *isPaused* v *CollaborationComponent.js* pričom sa táto premenná mení pomocou funkcie *pause()* a *unpause()*.

Panel kolaborantov

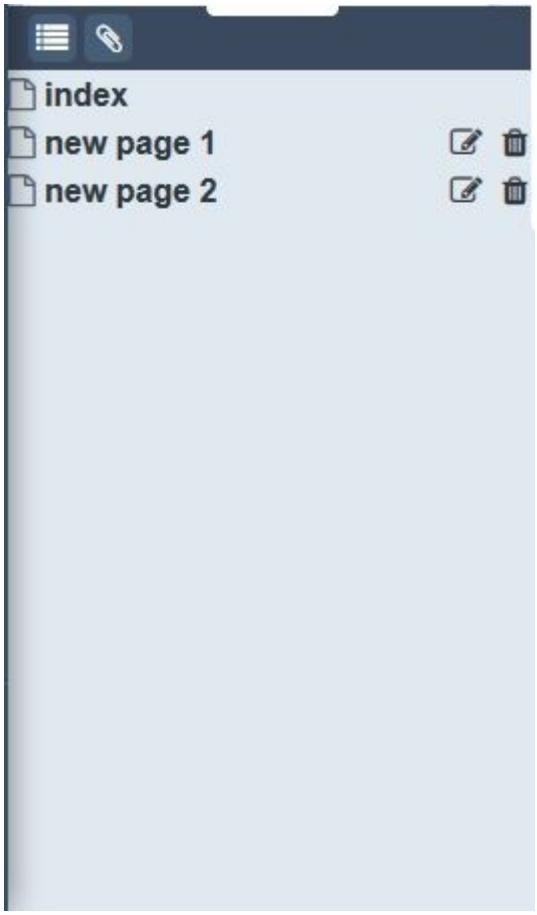
Pri prepínaní medzi stránkami sa každý kolaborant zobrazuje na momentálne aktívnej stránke alebo v inaktívnej zložke v prípade, že nie sú pripojení. V zátvorke sa zobrazuje počet kolaborantov a tieto zložky je možné otvárať alebo zatvárať.



Obrázok 11 - Obrazovka pre panel kolaborantov

Panel stránok

V paneli stránok sa zobrazujú všetky vytvorené stránky spolu s základnou index stránkou. Tieto stránky je možné premenovať pomocou tlačidla s ikonou ceruzky a vymazať s tlačidlom s ikonou odpadkového koša.



Obrázok 12 - Obrazovka pre panel stránok

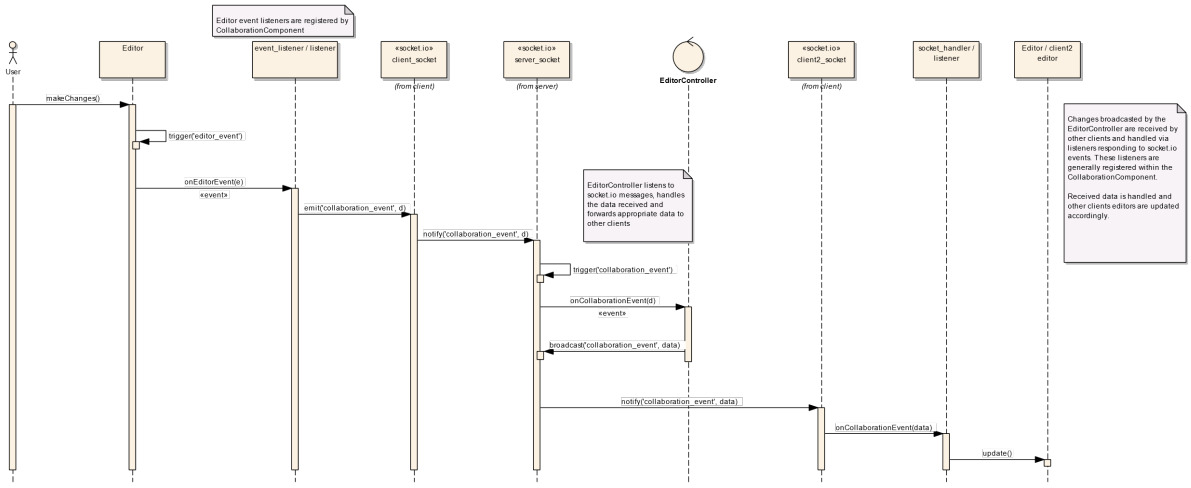
4.2.4 Synchronizácia zmien v prototypoch

Táto kapitola poskytuje vysokoúrovňový pohľad na zabezpečenie synchronizácie dát medzi používateľmi modifikujúcimi jeden prototyp používateľského rozhrania súčasne. Propagácia zmien vykonaných jedným klientom je zabezpečená dvoma aplikačnými komponentami:

- CollaborationComponent: komponent zabezpečujúci klientsku časť kolaborácie
- EditorController: ovládač na strane Node.js serveru zodpovedný za spracovanie dát od klientov vykonávajúcich zmeny a preposielanie spracovaných zmien ostatným klientom

Preposielanie dát po sieti je zabezpečené pomocou JavaScript knižnice socket.io. Vo všeobecnosti sú zmeny synchronizované nasledovným spôsobom:

1. Klient A vykoná zmeny v editore
2. Objekt editoru vyvolá patričnú udalosť
3. Listener zaregistrovaný CollaborationComponent inštanciou spracuje danú udalosť a potrebné dáta odošle na Node.js server pomocou socket.io
4. Server spracuje prijaté dáta a pomocou socket.io ich vyšle ostatným klientom pripojených do miestnosti stránky, na ktorej Klient A pracoval
5. Klienti B a C prijmú dáta zo serveru pomocou litenerov zaregistrovaných inštanciou CollaborationComponent na ich socket
6. Listener spracuje prijaté dáta a vykoná potrebné zmeny v editore



Obrázok 13 - Posielanie zmien vykonaných v editore ostatným klientom (všeobecný prípad)

Klient

Kolaborácia je na strane klienta zabezpečená pomocou komponentu CollaborationComponent. Zmeny vykonané v editore sú odosielané na server pomocou knižnice socket.io

CollaborationComponent.js

Komponent zabezpečujúci kolaboráciu na strane klienta. Okrem iného je zodpovedný predovšetkým za nasledovné:

- Registráciu listenerov pre udalosti, ktoré nastávajú počas kolaborácie v editore
- Registráciu socket.io listenerov, ktoré spracovávajú dáta prijaté zo serveru (socket.io namespace: *editor*)
- Spracovanie dát prijatých zo serveru
- Odosielanie lokálnych zmien server

initPlugin(client, editor, options)

Funkcia pre inicializáciu komponentu. Očakávané argumenty:

- *client:* inštancia knockout klienta
- *editor:* inštancia grapes.js editoru
- *options:* objekt s nastaveniami komponentu
    - *socket:* inštancia socket.io socketu pripojeného na *editor* namespace
    - *isPaused:* boolean, pokiaľ je atribút nastavený na true, tak sa komponent inštanciu v stave, v ktorom nebude reagovať na grapes.js a socket.io udalosti

Úlohou tejto funkcie je inicializovať komponent tak, aby bol schopný zabezpečiť kolaboráciu. Zavolaním tejto funkcie prvý krát sa zaregistrujú všetky potrebné event listenery pre grapes.js a socket.io a preťaží sa istá časť grapes.js funkcionality.

```
CollaborationComponent.prototype.initPlugin = function (client, editor, options) {
    var self = this, socket = undefined;

    if (this.plugin || !options.hasOwnProperty('socket')) {
        return;
    }

    socket = options.socket;

    this.instance = editor;
    this.client = client;
    this.isPaused = options.paused || false;

    grapesjs.plugins.add(this.pluginId, function (editor, options) {
        self.overrideEditorFunctionality(editor);
        self.enableRemoteAdding(editor, socket);
        self.enableRemoteRemoving(editor, socket);
        self.enableRemoteSelecting(self.client, editor, socket);
        self.enableRemoteUpdating(editor, socket);
        self.enableRemoteMoving(editor, socket);

        socket.on('component:assignIDs', function (data) {
            var source = undefined, target = undefined;

            if (typeof data === 'string' || data instanceof String) {
                target = editor.DomComponents.getComponents().models.slice(0);
                source = JSON.parse(data);
            } else {
                target = self.findComponent(data.cid);
                source = JSON.parse(data.JSONmodel);
            }

            if (target) {
                self.assignIDsToObjects(source, target);
                self.prepareCssSelectors(target);
            }
        });

        console.log('Collaboration plugin enabled!');
    });

    this.plugin = grapesjs.plugins.get(this.pluginId);
    if (this.plugin) {
        this.plugin(editor, options);
    } else {
        console.log('Could not initialize collaboration component.');
    }
};
```

overrideEditorFunctionality(editor)

Funkcia zodpovedná za úpravu správania editoru. V súčasnosti iba zabezpečuje rozšírenie konštruktorov grapes.js objektov.

```
CollaborationComponent.prototype.overrideEditorFunctionality = function (editor) {
    this.extendComponents(editor);
};
```

extendComponents(editor)

Funkcia pre rozšírenie konštruktorov objektov v inštancii grapes.js editoru. V rámci tejto funkcie sú rozšírené nasledovné konštruktory:

- kolekcia *Components*
- všetky konštruktory komponentov (objekty dediace od grapes.js triedy *Component)*

```
CollaborationComponent.prototype.extendComponents = function (editor) {
    //Override the default Components collection to allow listeneing to Component
add/remove events
    var Components = editor.DomComponents.Components, self = this;

    editor.DomComponents.Components = Components.extend({
        initialize: function (models, opts = {}) {
            Components.prototype.initialize.apply(this, arguments);
            this.on('add', self.onAddedToCollection);
        }
    });

    // Get the default models of all components and extend them with custom
functionality
    var componentTypes = editor.DomComponents.componentTypes, i = 0;
    for (i; i < componentTypes.length; i++) {
        componentTypes[i].model = this.extendDefaultComponent(editor,
componentTypes[i].model);
    }

    // Further extend other component models
    var linkCmpModel = this.getComponentType(editor, 'link').model;
    this.extendLinkComponent(editor, linkCmpModel);
};
```

Grapes.js v súčasnosti neposkytuje vhodný spôsob pre sledovanie pohybu komponentov pri presúvaní. Z toho dôvodu musela byť funkcia *initialize* kolekcie *Components* preťažená tak, aby sa dalo odsledovať, do ktorej kolekcie bol komponent vložený.

***Poznámka**: Presun komponentu **z** kolekcie sa dá odsledovať počúvaním udalosti run:tlb-move:before, ktorú vysiela inštancia editoru.*

Súvisiace časti: enableRemoteMoving(editor, socket)

extendDefaultComponent(editor, model)

Funkcia pre rozšírenie komponentov grapes.js editoru. Úlohou tejto funkcie je rozšíriť komponenty tak, aby:

- v JSON reprezentácii objektu neostávali dáta používané pre zabezpečenie kolaborácie
- aby sa dali pri inicializácií aplikovať štýly uložené v internom atribúte *__style*, do ktorého sa zároveň cacheujú (aby sa mohli odoslať ostatným klientom)

```
CollaborationComponent.prototype.extendDefaultComponent = function (editor, model)
{
    return model.extend({
        toJSON: function () {
            var obj = model.prototype.toJSON.apply(this, arguments);

            delete obj.addedByAnother;
            delete obj.addedLocally;
            delete obj.removedByAnother;

            return obj;
        },
        initialize: function (attr = {}, ops = {}) {

            if (this.get('addedByAnother') === true) {
                this.set('style', this.get('__style'), {silent: true});
                this.unset('__style', {silent: true});
            } else {
                this.set('__style', this.get('style'), {silent: true});
            }

            model.prototype.initialize.apply(this, arguments);
        },
        initComponents: function () {
            // Have to add components after the init, otherwise the parent
            // is not visible
            const comps = new editor.DomComponents.Components(null, this.opt);
            comps.parent = this;
            !this.opt.avoidChildren && comps.reset(this.get('components'));
            this.set('components', comps);
            return this;
        }
    });
};
```

findComponent(cid, [byRemote = False]

Funkcia pre hľadanie komponentov v grapes.js strome. Pokiaľ je atribút *byRemote* false, tak budú komponenty vyhľadávané pomocou atribútu *cid*, ktorý je interne generovaný grapes.js. Tento atribút identifikuje komponenty iba lokálne a môže sa u rôznych klientov líšiť a navyše je generovaný pri každom inštanciovaní komponentu – tzn., že pri každej relácii môže mať inú hodnotu.
Pokiaľ je hodnota argumentu *byRemote* true, tak budú komponenty hľadané pomocou interného atribútu *__sharedID.* Tento atribút je generovaný serverom a mal by správne identifikovať komponenty u všetkých klientov.

assignIDsToObjects(source, target, [idAttr = "__sharedID"])

Funkcia pre priraďovanie ID generovaných serverom lokálnym komponentom. Táto funkcia je volaná napríklad v prípade, že klient A pridá nový komponent do prototypu stránky. Nový komponent (a jeho deti) sa odošlú na server, ktorý im všetkým vygeneruje unikátne id. Toto id sa uloží do atribútu *__sharedID* a upravené komponenty sa odošlú ostatným klientom. Následne sa aktualizovaný strom odošle aj klientovi A, u ktorého sa novo-vygenerované id priradia vytvoreným komponentom.

enableRemoteAdding(editor, socket)

Funkcia inicializujúca synchronizáciu vytvárania nových komponentov. Táto funkcia zaregistruje listener na udalosť *component:add* grapes.js editoru, ako aj rovnomenný listener pre klientsky socket. Vyvolanie udalosti *component:add* editorom znamená, že bol pridaný nový komponent, ktorý sa musí odoslať na server. Vyvolanie udalosti *component:add* socketom znamená, že bol zo serveru prijatý nový komponent, ktorý sa musí pridať do editoru.

***Poznámka**: Presúvanie komponentu v editore z jedného miesta na druhé v súčasnosti nespustí udalosť component:add. Napriek tomu ale stále spustí udalosť component:remove.*

(editor, socket)

Funkcia inicializujúca synchronizáciu vymazávania nových komponentov. Táto funkcia zaregistruje listener na udalosť *component:remove* grapes.js editoru, ako aj rovnomenný listener pre klientsky socket. Vyvolanie udalosti *component:remove* editorom znamená, že bol odstránený komponent. O tejto skutočnosti je server informovaný pomocou socket.io správy označenej *component:remove.* Prijatie udalosti *component:remove* socketom znamená, že iný klient vymazal komponent, ktorý treba odstrániť lokálne.

***Poznámka**: Presúvanie komponentu v editore z jedného miesta na druhé v súčasnosti nespustí udalosť component:add. Napriek tomu ale stále spustí udalosť component:remove.*

enableRemoteUpdating(editor, socket)

Funkcia inicializujúca synchronizáciu zmien vykonaných nad komponentami. Táto funkcia registruje listenery na nasledovné grapes.js udalosti:

- *component:styleUpdate* – udalosť vyvolaná pri zmene štýlu komponentu
- *component:update* – udalosť vyvolaná pri zmene atribútov komponentu

Táto funkcia registruje listenery na nasledovné socket.io udalosti:

- *component:styleUpdate* – udalosť vyvolaná pri prijatí zmien štýlu komponentu od serveru
- *component:update* – udalosť vyvolaná pri prijatí zmien atribútov komponentu od serveru

Táto funkcia ďalej registruje *add* listener na kolekciu CSS pravidiel grapes.js komponentu *CssComposer*, ktorý je súčasťou inštancie editora.

enableRemoteMoving(editor, socket)

Funkcia zabezpečujúca synchronizáciu presúvania komponentov. Úlohou tejto funkcie je zaregistrovať listenery na grapes.js udalosť *run:tlb-move:before.* Táto udalosť je vyvolaná keď používateľ začne presúvať komponent v editore. Ďalej táto funkcia obsahuje listener, ktorý sa spustí pri pridaní komponentu do *Components* kolekcie. Tento stav nastane pri vytvorení nového komponentu, ale aj pri premiestnení komponentu z jedného miesta na druhé. Tieto dva stavy sa rozlíšia a informácie o presunutom komponente sa odošlú na server. Informácie o komponentoch presunutých ostatnými klientami sú prijaté pomocou socket.io socketu a správy označenej *component:moved*.

***Poznámka**: Presúvanie komponentu v editore z jedného miesta na druhé v súčasnosti nespustí udalosť component:add. Napriek tomu ale stále spustí udalosť component:remove.*

Súvisiace časti: extendComponents(editor)

socket.io udalosti

Táto časť obsahuje opis socket.io udalostí, ktoré môže klient obdržať v súvislosti s procesom zabezpečenia kolaborácie viacerých klientov.

component:add

Správy obsahujúce nový komponent, ktorý bol pridaný iným klientom.

component:assignIDs

Správy obsahujúce vygenerované ID, ktoré treba priradiť lokálnym komponent aby bolo možné identifikovať komponenty počas kolaborácie.

component:remove

Informácie o vymazaní komponentu iným klientom.

component:update

Zmena atribútov komponentu.
component:styleUpdate

Zmena štýlu komponentu.

component:moved

Zmena pozície komponentu.

Server

Zodpovednosť serveru v procese kolaborácie leží prioritne v preposielaní dát medzi klientami a ukladanie dát do databázy Cassandra.

EditorController.js

Ovládač zodpovedný za zabezpečenie preposielania dát medzi klientami pomocou socket.io socketu pre *editor* namespace.

onConnect(socket, connection)

Funkcia volaná pri úspešnom pripojení klienta na server.
Táto funkcia plní nasledovné úlohy:

- priradí socket do miestnosti projektu
- pridá socket do miestnosti stránky projektu
- zaregistruje listenery prístupné používateľom s oprávnením *watch*
- zaregistruje listenery prístupné používateľom s oprávnením *edit*

joinPageRoom(socket)

Pridanie socketu do miestnosti projektovej stránky. Pokiaľ je socket pripojený do inej miestnosti stránky, tak bude z tejto miestnosti vyradený.

joinProjectRoom(socket)

Pridanie socketu do miestnosti projektu.

addWatchLevelListeners(socket)

Registrácia listenerov pre udalosti prístupné používateľom s oprávnením *watch*. Napr.: zmena stránky, načítanie dát prototypu a načítanie komponentov, s ktorými pracujú ostatní kolaboranti.

addEditLevelListeners(socket)

Pridanie listenerov pre udalosti prístupné používateľom s oprávnením *edit:*

- ukladanie dát stránky
- voľba komponentu
- zrušenie voľby komponentu
- uzamknutie/odomknutie komponentu
- pridanie, odstránenie, presunutie a úprava komponentu

onComponentAdd(data, socket)

Funkcia pre spracovanie novo pridaný komponentov. Táto funkcia prijme dáta stromu nových komponentov, priradí im ID, odošle komponenty ostatným kolaborantom na pridanie a odošle upravený strom jeho tvorcovi pre priradenie nových ID.

```
EditorController.prototype.onComponentAdd = function (data, socket, respond) {
    var self = this;
    var page_id = socket.editorData.page_id;

    //Synchronous query to prevent counter value collision during multiple requests
    Page.findOne({page_id: page_id}, function (err, page) {
        if (err || !page) return;
        var component = JSON.parse(data.JSONmodel);
        var generated = CM.prepareComponents(component, page.component_counter,
true);

        component = generated.root;
        data.JSONmodel = JSON.stringify(component);

        console.log('Saving page', page.page_id, 'with counter',
page.component_counter);
        Page.update({page_id: socket.editorData.page_id}, {component_counter:
Long.fromInt(generated.counter)}, {}, function (err) {
            if (err) {
                console.log(err);
            } else {
                socket.emit('component:assignIDs', data);
                socket.broadcast.to(socket.pageRoom).emit(
                    'component:add',
                    self.withUser(data, socket)
                );
            }
        });
    });
};
```

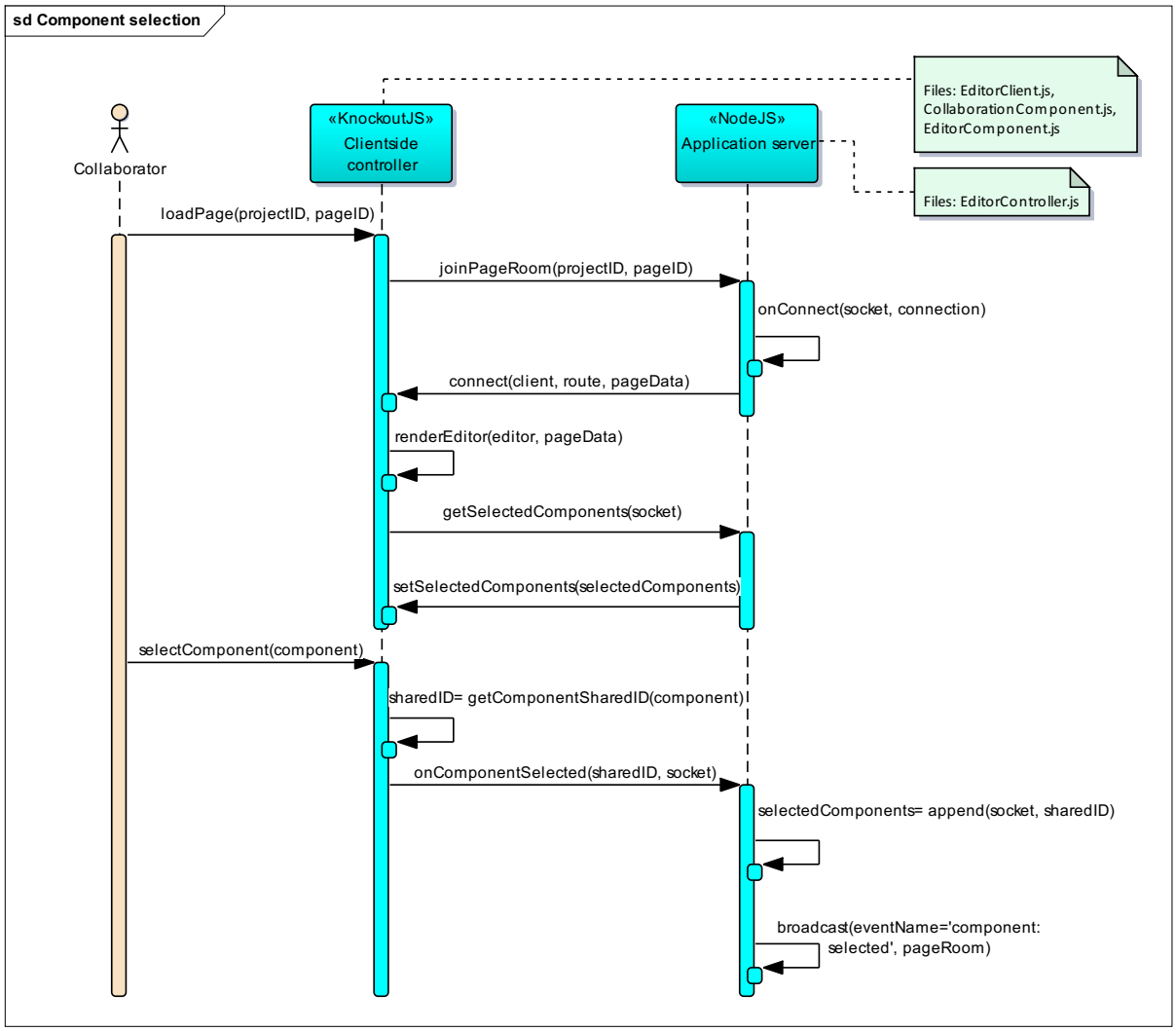
onComponentUpdate, onComponentStyleUpdate, onComponentRemove, onComponentMoved

V kontexte kolaborácie je úlohou tejto sady funkcii prakticky iba preposlať dáta ostatným kolaborantom.

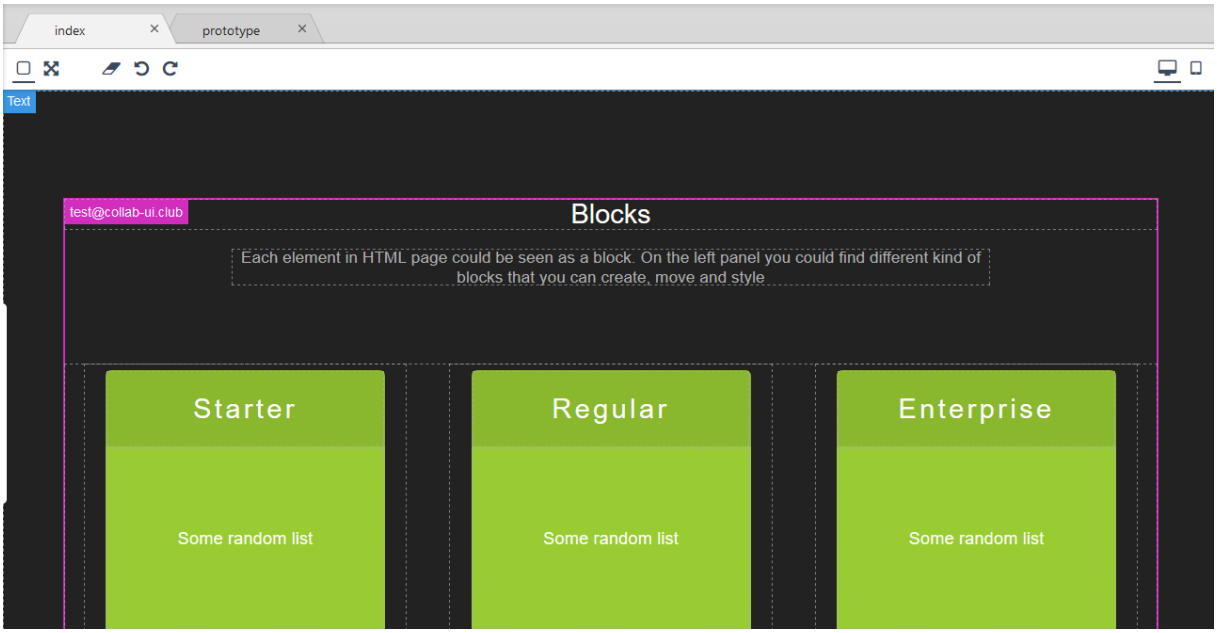
4.2.5 Vizualizácia označených komponentov

Zvýrazňovanie označených komponentov je pridaná funkcionalita nad interaktívnym editorom aplikácie, ktorý vychádza z knižnice GrapesJS. Vďaka nej má používateľ vždy prehľad o tom, s akými komponentami pracujú ostatní členovia tímu. Zmeny sú uložené na aplikačnom serveri a prenášajú sa vždy pri pripojení alebo odpojení kolaboranta alebo pri označení ľubovoľného komponentu. Zároveň pri zmene šírky bočných panelov editora vždy dochádza ku explicitnému prekresleniu vyznačených komponentov, nakoľko v opačnom prípade by mohlo dôjsť k ich nesprávnemu zarovnaniu.

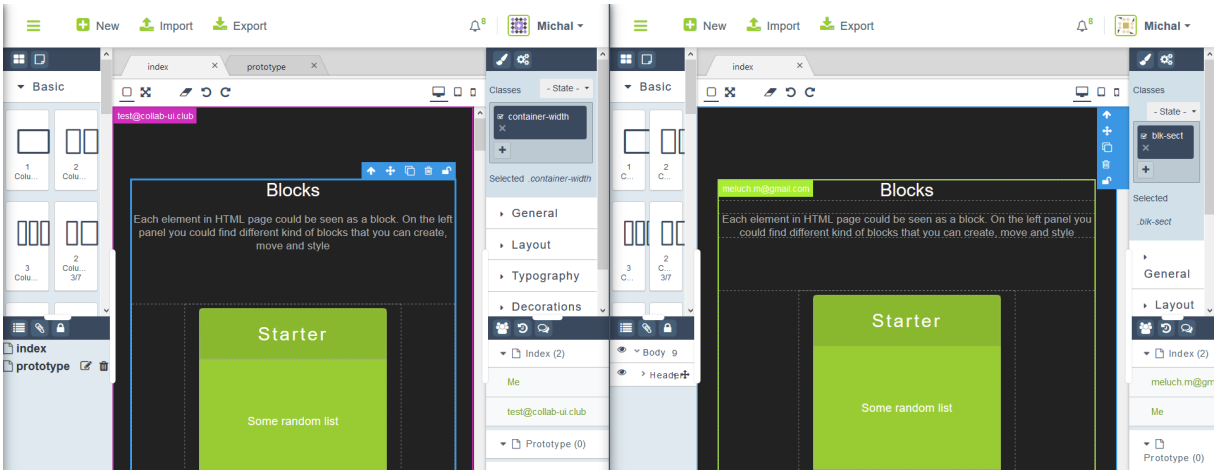
Pri zvýrazňovaní dôjde ku prepisu CSS štýlu daného komponentu (pridanie orámovania) a ku pridaniu HTML elementu obsahujúceho štítok s e-mailovou adresou daného kolaboranta. Nemodifikujeme teda funkcionalitu knižnice tretej strany, ale pridávame separátne akcie, ktoré sa vykonajú až po skončení všetkých ďalších udalostí.



Obrázok 14 - Udalosti súvisiace s označením komponentu v editore po jeho načítaní



Obrázok 15 - Zvýraznenie komponentu, ktorý je práve označený iným kolaborantom



Obrázok 16 - Zvýraznenie označených komponentov medzi dvoma rôznymi kolaborantami súčasne

Označovanie komponentov prebieha tak, ako všetky ostatné akcie pomocou websocketov, ktoré obsluhuje aplikačný server NodeJS. Po každom označení komponentu na strane klienta dôjde ku vyslaniu signálu `component:selected` a jeho obslúženiu metódou `onComponentSelected` na strane servera (`EditorController.js`).

Odoslanie udalosti vlastného označenia komponentu

Táto časť opisuje implementáciu súvisiacu s označením komponentu na strane klienta a prenesením tejto zmeny ku ostatným kolaborantom. Server si udržiava zoznam označených komponentov v globálnej premennej `selectedComponents`. Ide o objekt, ktorý si uchováva informácie o tom, ktorý označený komponent patrí ku ktorej stránke (resp. prototypu) a kto je jeho „majiteľom" – t. j. kto ho označil:

```
selectedComponents = {
      pageRoom: {
            userToken1: { selectedComponentID },
            userToken2: { selectedComponentID },
            ...
      }
}
```

Priebežné uchovávanie označených komponentov v serverovej premennej je dôležité pre prípad, že sa k projektu pripojí ďalší kolaborant a chceme mu poskytnúť informáciu o tom, ktoré komponenty sú na danej stránke už označené inými ľuďmi. Na novoprijatú správu o označení komponentu zareaguje tak, že si k danej stránke priradí nový záznam o označení komponentu kolaborantom. Následne rozošle informáciu o tejto udalosti všetkým ostatným kolaborantom pracujúcim na danej stránke (resp. prototype). Informácie o pôvodcovi udalosti (`userToken`, `pageID`) berie server z objektu socketu, ktorý sa defaultne posiela spolu s údajmi a názvom udalosti pri akejkoľvek komunikácii:

```
socket.on('component:selected', function (data) {
    self.onComponentSelected(data, socket);
});
```

```
EditorController.prototype.onComponentSelected = function (data, socket) {
    var self = this;

    // If this is the first user selecting a component in the page room
    if (!this.selectedComponents.hasOwnProperty(socket.pageRoom)) {
        this.selectedComponents[socket.pageRoom] = {};
    }

    var component = {
        'idPrev': self.selectedComponents[socket.pageRoom][socket.userToken],
        'idNew': data
    };

    // Notify all other users in the page room about the newly selected component
    socket.broadcast.to(socket.pageRoom).emit(
        'component:selected',
        self.withUser(component, socket)
    );

    // Update the server-side selected components object
    this.selectedComponents[socket.pageRoom][socket.userToken] = data;
};
```

Implementácia odoslania udalosti označenia komponentu na strane klienta je pomerne jednoduchá a zahŕňa len vyhľadanie jeho zdieľaného ID (špeciálna vetva pre body element) a odoslanie údajov na server:

```
// Notify the server when the current user selects a component (editor listener)
editor.on('component:selected', function (components) {

    var changedComponent = components[0].changed.selectedComponent;
    var sharedID = undefined;
    self.mySelectedComponent = changedComponent;

    console.log('Selected component: ', changedComponent);

    // Special case - body (wrapper) component
    if (self.isComponentWrapper(changedComponent)) {
        sharedID = changedComponent.cid;
    }
    // All other user components
    else if (changedComponent.attributes.hasOwnProperty('__sharedID')) {
        sharedID = changedComponent.attributes['__sharedID'];
    }
    else {
        console.error('Failed to retrieve selected component ID from: ', changedComponent);
        return;
    }

    socket.emit('component:selected', sharedID);
});
```

Odznačenie komponentu je implementované obdobne, avšak iniciátorom udalosti je server. Dôvodom je, že udalosť odznačenia komponentu sa odosiela len pri odpojení kolaboranta z aktuálnej stránky prototypu (napr. vypnutie prehliadača, zmena aktuálneho prototypu..). Informáciu o tejto udalosti má k dispozícii logicky len server – konkrétne je iniciátorom tohto volania funkcia `EditorController.onDisconnect(socket)`.

```
/**
 * Broadcasts a notice that the user with defined socket has deselected a component
 *
 * @param socket
 */
EditorController.prototype.emitComponentDeselect = function (socket) {
    var self = this;

    if (this.hasUserAnyElementSelected(socket.pageRoom, socket.userToken)) {
        socket.broadcast.to(socket.pageRoom).emit(
            'component:deselected',
            {
                user: {'token': socket.userToken},
                component: {'id': self.selectedComponents[socket.pageRoom][socket.userToken]}
            }
        );

        // Also remove the component from the selected components object
        delete this.selectedComponents[socket.pageRoom][socket.userToken];
    }
};
```

Prijatie udalosti cudzieho o(d)značenia komponentu

Táto časť opisuje implementáciu scenára, pri ktorom používateľ pracujúci na prototype obdrží informáciu o označení alebo odznačení komponentu iným kolaborantom, ktorú mu odoslal server. Okamžite po prijatí tejto informácie na strane klienta dôjde ku odznačeniu komponentu, ktorý mal auto udalosti pôvodne označený (metóda `remSelectedComponentStyle`). Následne je na základe informácie o pôvodcovi udalosti a čísla cieľového komponentu možné ho označiť:

```
// Respond to servers notification about another user (or me on other devices/tabs) selecting a component
socket.on('component:selected', function (data) {

    var userToken = data.user.token;
    var user = self.getUserByToken(client, userToken);

    // Reset any previously selected components for this user
    self.remSelectedComponentStyle(userToken);

    self.addSelectedComponentStyle(
        data.component.idNew,
        user.email,
        user.token
    );
});
```

```
// Highlight a component selected by another collaborator
CollaborationComponent.prototype.addSelectedComponentStyle = function (sharedID, userEmail, userToken) {
    var self = this;
    var component = self.findComponent(sharedID, true);

    // Ignore unitialized components (e. g. 'uncommited' text nodes)
    if (!component || !component.view) {
        return;
    }

    // Convert the GrapesJS component to a jQuery html DOM object
    var componentHtml = $(component.view.el);

    // Generate the user email badge element & random color to be used for this user
    var selectedComponentBadge = $('<div id="badge' + userToken.replace('@', '') + '">' + userEmail + '</div>')
    var userColor = randomColor({'seed': userToken, 'luminosity': 'bright'});
    var outlineSize = 2;  // px

    // Set the user email badge dynamic styles & position
    selectedComponentBadge.addClass('gjs-badge user-selection user-selection-badge');
    selectedComponentBadge.css({···
    });

    // Set the selected element outline style
    componentHtml.addClass('user-selection-outline')
    componentHtml.css({···
    });

    $('#gjs-tools').prepend(selectedComponentBadge);

    // The canvas wrapper doesn't get a shared ID - send its base CID instead
    this.selectedComponents[userToken] = {
        'id': self.isComponentWrapper(component) ? component.cid : component.attributes['__sharedID'],
        'email': userEmail
    };
};
```

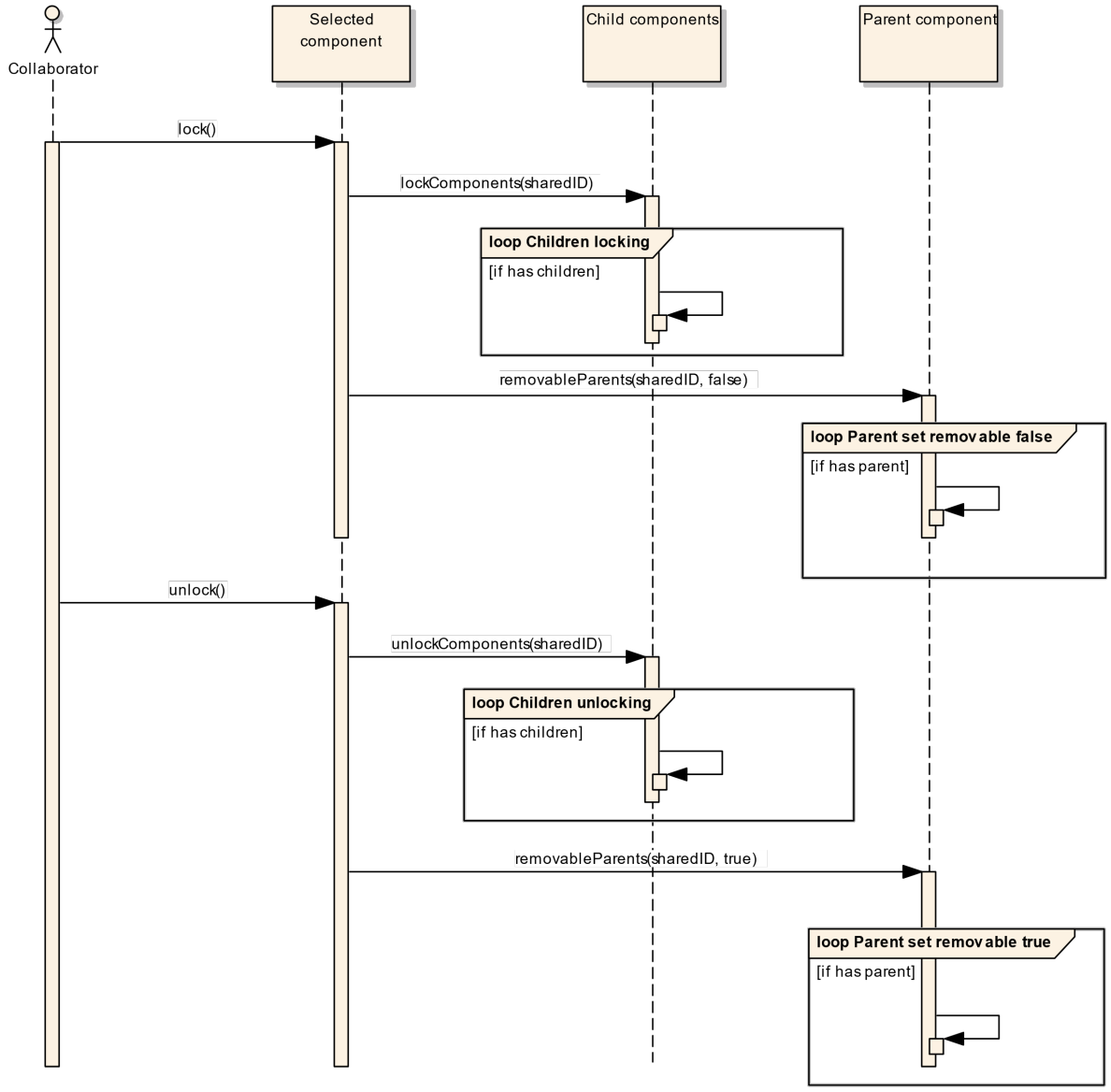
Označenie komponentu prebieha tak, že sa k jeho CSS štýlu pridá atribút outline a do editora sa vygeneruje nový HTML element štítku s e-mailom kolaboranta, ktorý ho označil. Farba tohto štítku, ako aj orámovanie (outline) cieľového komponentu je generovaná pseudonáhodne na základe e-mailu kolaboranta, ktorý ho označil.

Informácie o označených komponentoch sú po prijatí uchovávané na strane klienta (`CollaborationComponent.selectedComponents`) z dôvodu, že je nutné ich prekreslenie vždy po zmene veľkosti bočných panelov a frekvencia týchto volaní by zbytočne zaťažovala server.
Odznačenie komponentu je na strane klienta spracované obdobne ako označenie – t. j. dôjde ku odstráneniu štítku s menom kolaboranta a orámovania cieľového komponentu. Na záver sa odstráni aj záznam o tomto komponente z lokálneho objektu označených komponentov:

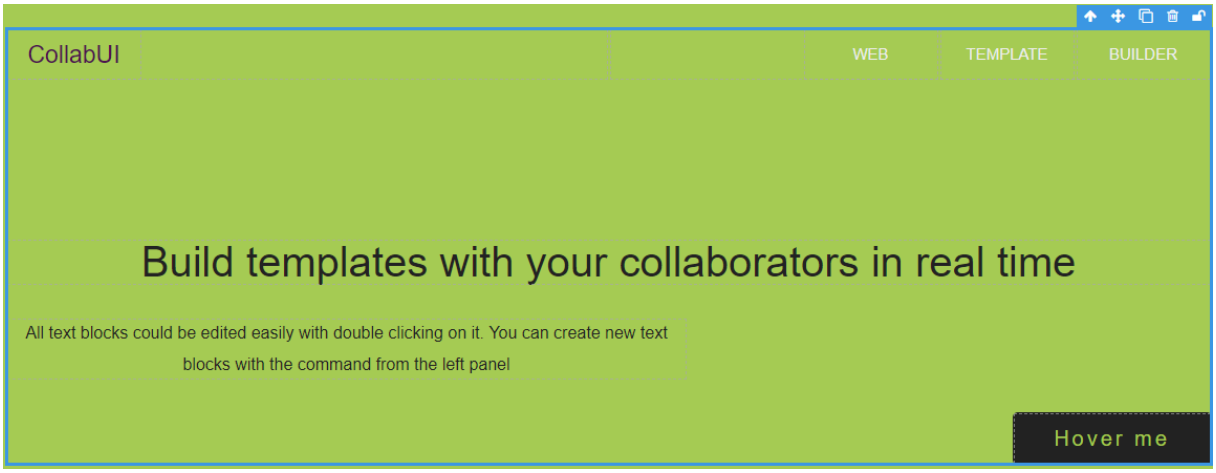
```
CollaborationComponent.prototype.remSelectedComponentStyle = function (userToken) {
    // Remove the generated user email badge on the unselected component
    $('div#badge' + userToken.replace('@', '')).remove();

    // The selected component style has been already removed (I'm the one who deleted it, for instance)
    if (!this.selectedComponents[userToken]) {
        return;
    }

    // Retrieve the unselected component
    var unselectedComponent = this.findComponent(
        this.selectedComponents[userToken].id, true
    );

    if (unselectedComponent) {
        // Cast the unselected component HTML to a jQuery object
        var unselectedComponentHtml = unselectedComponent.view.el;
        unselectedComponentHtml = $(unselectedComponentHtml);

        // Remove the outline effect
        unselectedComponentHtml.css({
            'outline': 'initial',
            'outline-color': 'initial'
        });
    }

    delete this.selectedComponents[userToken];
};
```

4.2.6 Uzamykanie komponentov

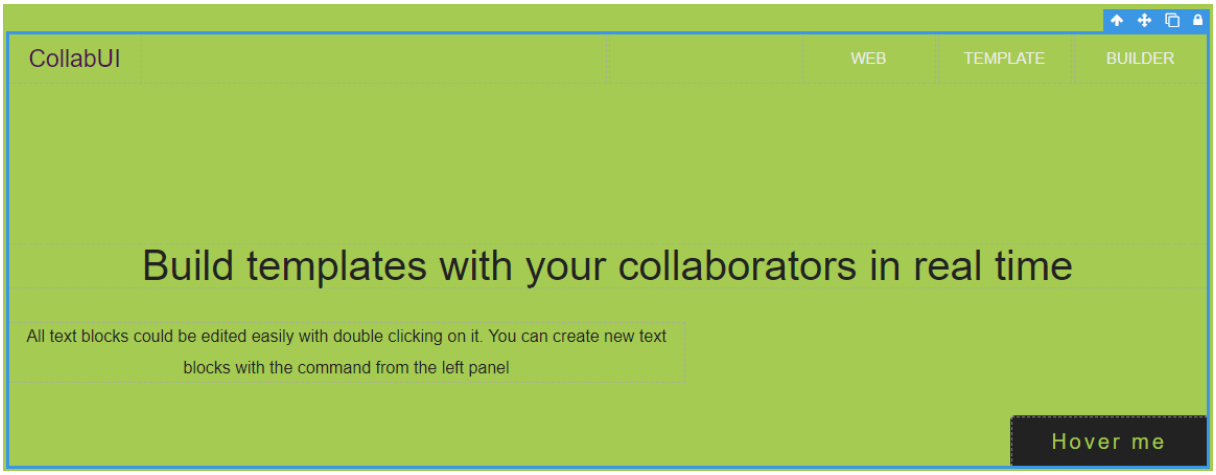
Uzamykanie komponentov je rozširujúca funkcionalita pre GrapesJS. Pomocou uzamykania si môže kolaborant uzamknúť vybraný komponent, ktorý nebudú môcť ostatní kolaboranti upravovať ani premiestňovať. Pri uzamknutí komponentu sa uzamknú aj deti vybraného komponentu. Rovnako sa musí zakázať vymazávanie pre všetky rodičovské komponenty uzamknutého komponentu. Ak by sa vymazávanie rodičovských komponentov nenastavilo, tak by sa mohlo stať, že niekto vymaže rodiča aj s uzamknutým komponentom. Uzamknúť komponent sa dá pomocou ikony zámku v tooltipe označeného komponentu viď. obrázok 2. Odomknúť komponent sa dá pomocou ikony odomknutého zámku v tooltipe označeného komponentu viď. obrázok 3 alebo pomocou zoznamu uzamknutých komponentov v panely viď. obrázok 4. Do zoznamu uzamknutých komponentov sa nepridávajú komponenty uzamknuté ostatnými kolaborantmi. Taktiež sa komponenty odomknú keď kolaborant, ktorý komponenty uzamkol opustí konkrétnu stránku projektu resp. opustí socket.



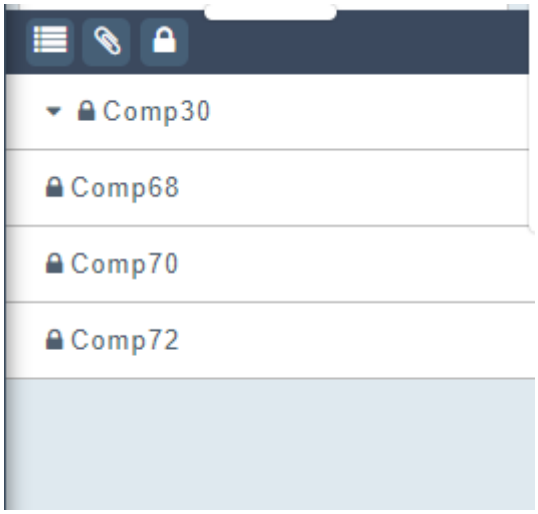
Obrázok 17 - Uzamykanie/odomykanie komponentov



Obrázok 18 - Odomknutý komponent



Obrázok 19 - Uzamknutý komponent



Obrázok 20 - Zoznam uzamknutých komponentov

Uzamykanie a odomykanie v rámci kolaborácie prebieha pomocou soketov. Na node servery (EditorController.js) prebieha spracovanie požiadavky od kolaboranta, ktorý komponent uzamkol alebo odomkol. Následne je pomocou soketov táto požiadavka rozposlaná medzi pripojených kolaborantov. Pre lepšie pochopenie implementácie je potrebné uvedomiť si, či uzamykanie/odomykanie prebieha u kolaboranta, ktorý túto požiadavku inicioval alebo, či sa jedná o kolaboranta, ktorý túto požiadavku prijal.

Prvým krokom pri implementácií je pridanie tlačidla do tooltipu označeného komponentu (CollaborationComponent.js):

```
tb.push({
  id:                                                    'locker',
  attributes:          {class:          'fa          fa-lock          locker'},
  command:                                               'locking'
});
```

Tlačidlo obsahuje príkaz locking, ktorý pracuje s GrapesJS pričom je potrebné tento príkaz najprv pridať aj pri inicializácií GrapesJS inštancie (EditorComponent.js):

```
commands: {
  defaults: [  {
    id: 'locking',
    run:  function(editor, sender){

    },
    stop:  function(editor, sender){

    },
  }
   ],
```

Ak takto definujeme príkaz pomocou GrapesJS, tak je teraz možné na tento príkaz počúvať:

```
editor.on('run:locking', function () {})
```

Každý kolaborant má uložený zoznam uzamknutých komponentov (lockedComponents v CollaborationComponent.js) s ktorým sa počas všetkých procesov uzamykania a odomykania pracuje. Taktiež na servery (lockedComponents v EditorController.js) existuje rovnaký zoznam uzamknutých komponentov ale s tým rozdielom, že obsahuje všetky uzamknuté komponenty naprieč všetkými existujúcimi projektami aby sa po pripojení na konkrétny projekt poslali všetky aktuálne uzamknuté komponenty kolaborantovi.

Uzamykanie

Po kliknutí na ikonu zámku obrázok 2 sa spustí príkaz run:locking. Po spustení príkazu sa pridajú do zoznamu uzamknutých elementov aktuálne zvolený komponent a všetky jeho deti, funkcia lockComponents, ktorá vracia zoznam všetkých zamknutých komponentov. V rámci funkcie lockComponents sa ešte zavolá funkcia removableParents, ktorá všetkým rodičovským komponentom pozastaví možnosť vymazania.

```
var components = self.lockComponents(sharedID);

socket.emit('component:locked', components);
```

Zoznam uzamknutých komponentov je následne poslaný cez socket na server (EditorCOmponent.js), ktorý počúva na tento emit.

```
socket.on('component:locked', function (data) {
   self.onComponentLocked(data, socket);
});
```

Po zachytení na strane server sa zavolá funkcia onComponentLocked, ktorá pridá uzamknuté komponenty do zoznamu uzamknutých komponentov na strane server a ďalej broadcastne zoznam uzamknutých komponentov ďalším kolaborantom pripojeným na ten istý projekt.

```
socket.broadcast.to(socket.pageRoom).emit(
   'component:locked',self.withUser(data, socket, "components")
);
```

Následne všetci ostatní kolaboranti počúvajú na tento emit (CollaborationComponent.js).

```
socket.on('component:locked', function (data) {})
```

Po zachytení tohoto emitu sa každému kolaborantovi pridajú uzamknuté komponenty to jeho vlastného zoznamu uzamknutých komponentov. Následne je treba spraviť aj to aby sa uzamknuté komponenty priamo v editore nedali upravovať alebo s nimi hýbať pomocou funkcie remoteLocking.

Pridanie existujúcich uzamknutých komponentov po príchode kolaboranta zabezpečuje odoslanie emitu zo strany server.

```
socket.emit(
   'collaborators:set_locked_components', self.lockedComponents[socket.pageRoom]
);
```

A na strane klienta sa tento emit zachytí.

```
socket.on('collaborators:set_locked_components', function (data) {})
```

Odomykanie

Proces odomykania je v podstate rovnaký ako proces uzamykania. Kolaborant spustí funkciu unlockComponents (CollaborationComponent.js), čím odstráni zo svojho zoznamu odomknuté komponenty a tento zoznam odošle soketom na server.

```
var components = self.unlockComponents(sharedID);

socket.emit('component:unlocked', components);
```

Zoznam odomknutých komponentov je následne na servery odstránený zo zoznamu uzamknutých komponentov na servery (lockedComponents v EditorController.js). Server počúva na component:unlocked pričom následne zavolá funkciu onComponentUnlocked.

```
socket.on('component:unlocked', function (data) {
  self.onComponentUnlocked(data, socket);
});
```

Vo funkcii onComponentUnlocked server broadcastne ostatným kolaborantom zoznam odomknutých komponentov.

```
socket.broadcast.to(socket.pageRoom).emit(
  'component:unlocked', data
);
```

Následne všetci ostatní kolaboranti počúvajú na tento emit (CollaborationComponent.js).

```
socket.on('component:unlocked', function (data) {})
```

Po zachytení tohoto emitu sa každému kolaborantovi odstránia uzamknuté komponenty z jeho vlastného zoznamu uzamknutých komponentov. Následne je treba spraviť aj to aby sa odomknuté komponenty priamo v editore dali upravovať pomocou funkcie remoteLocking.

5 Testovanie

V rámci ladenia synchronizácie editora sme pristúpili k testovaniu jednotlivých preddefinovaných komponentov, pričom vzniklo mnoho testovacích scenárov, ktoré nájdete v priloženom súbore **testovanie_komponentov.zip**