Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 3, 842 16 Bratislava 4

Webový editor Texu - Webtex

Tímový projekt

Dokumentácia k inžinierskemu dielu

Tím č. 15 – code-X

Bc. Igor Hula

Bc. Kazimír Jaroszewicz

Bc. Radoslav Kontúr

Bc. Radovan Kuka

Vedúci tímového projektu: Ing. Tomáš Kramár
Akademický rok: 2011/2012

Obsah

Úvod

Účel a rozsah dokumentu

Témou nášho tímového projektu je Webový editor pre TeX. Na webe je dostupných len málo editorov podobného zamerania. Problémom mnohých je to, že nazhŕňajú všetko čo by bežný používateľ očakával. Preto sme existujúce riešenia podrobili z analýze z ktorej pre nás vyplynuli užitočné informácie, akých chýb sa vyvarovať a akú hodnotu pridať.

Cieľom nášho tímového projektu je vytvorenie interaktívneho webového editora, ktorý by umožňoval úpravu zdrojových kódov v TeXu (LaTeX, ConTeXt...), ich kompiláciu a prezeranie výsledného PDF súboru. Jadro funkcionality editora by malo spočívať v podpore kolaborácie. Editor by mal podporovať verziovanie dokumentu, sledovanie zmien, zadávanie poznámok a tým umožniť plnohodnotnú kolaboratívnu tvorbu dokumentov. Webový editor by mal umožňovať tvorbu dokumentu aj mimo prostredia webu a poskytovať jednoduchú obojsmernú synchronizáciu s lokálnymi súbormi. Súčasťou projektu je aj vytvorenie API, aby sa dal editor jednoducho zaintegrovať s niektorým z existujúcich desktopových editorov a rozšíriť ich o možnosť prezerania zmien a poznámok.

Riešenie projektu si vyžaduje širokú škálu zručností a technológií: HTML5&CSS3 (rozhranie); JavaScript, CofeeScript, jQuery, Backbone.js (pre klientsku časť), Ruby, Rails, Node.js a PostgreSQL pre serverovú časť.

Projekt riešime agilnou vývojovou metodikou Scrum s dôrazom na testami riadený vývoj, kde je vývoj rozdelený na dvojtýždňové úseky, tzv. šprinty. V rámci každého šprintu robíme analýzu, návrh, implementáciu a testovanie zvolených funkcionalít, ktoré sú opísané tzv. príbehmi používateľa. Je zvykom šprinty pre ľahšiu identifikáciu pomenúvať podľa nejakého kľúča, my sme si ženské mená zoradené abecedne.

Predkladaný dokument, ktorý predstavuje dokumentáciu k inžinierskemu dielu, odráža toto rozdelenie a zachytáva prácu v jednotlivých šprintoch počas zimného semestra akademického roku 2011/2012.

1 Prvý šprint - Alexandra

V prvom šprinte sme venovali štúdiu, pre nás nového, programovacieho jazyka, oboznamovaním sa s Rails frameworkom a prípravou prostredia pre aplikáciu a vývoj softvéru. Riešili sme nasledovné úlohy:

Úloha	Plnenie
Pridať projekt do Jenkinsa	Jenkins budeme používať na automatické nasadzovanie zmien do
Vytvoriť repozitár a skupinu na Gitbuse	na školskom Git serveri sme vytvorili skupinu Codex s repozitárom pre náš projekt WebTex, ktorý sa nachádza tu: http://gitbus.fiit.stuba.sk/+codex
Rozbehať vývojové prostredie	rozhodli sme sa pre Ubuntu 11.10 distribúciu, ktorú sme nainštalovali ako virtuálny počítač pod VirtualBox.
Nainštalovať a nakonfigurovať PostgreSQL	je nainštalovaný v rámci vývojového prostredia aj produkčného servera
Nainštalovať a nakonfigurovať passenger	nainštalovaný na serveri, slúži na jednoduché nasadzovanie aplikácie
Vygenerovať základnú kostru Rails aplikácie	vygenerovaná príkazom „rails new webtex“
Naštudovať ako funguje SYNCTEX	túto úlohu sme si nechali na neskorší šprint, kedy budeme riešiť zobrazovanie vygenerovaných PDF dokumentov. SyncTeX slúži na synchronizáciu medzi zdrojovým TeX dokumentom a vygenerovaným PDF dokumentom
Naštudovať ako funguje Git note	Naštudovali sme Git note a uvažovali sme výhody a možnosti použitia v našom rpojekte
Porovnať Codemirror a ACE	rozhodli sme sa pre CodeMirror na základe funkcií, ktoré poskytuje
Rozdistribuovať konvencie programovania pre Ruby a Git	rozhodli sme sa ísť podľa štýlu, ktorý nám navrhol vedúci a tento štýl je bližšie špecifikovaný v Dokumentácii k riadeniu

2 Druhý šprint - Božena

V druhom šprinte sme riešili nasledovné úlohy:

2.1 Autentifikácia používateľov

Názov user story	Popis
Autentifikácia používateľov	Ako používateľ sa chcem prihlásiť, aby som mohol mať privátne projekty, ktoré sú viditeľné len mne

V rámci úlohy sme riešili následovné podúlohy:

- Nainštalovať a naštudovať DEVISE
- Nakonfigurovať DEVISE
- Cucumber testy

2.1.1 Implementácia

Implementácia má obsahovať formulár, ktorým sa používateľ prihlási do systému. Mal by použiť kombináciu e-mailu a hesla. Pre autentifikáciu sme použili gem Devise. Je to modulárne autentifikačné riešenie pre Rails framework. Po inštalácií samotného gem-u sme vygenerovali model (users.rb), konfiguračný súbor a views pre všetky akcie. Model sme nakonfigurovali na ukladanie šifrovaných hesiel do databázy a validáciu používateľov, registráciu, zapamätanie si používateľov cez cookies, sledovanie prihlásení a obnovenie hesla. Samotný zdrojový kód vyzerá nasledovne:

`Config/initializers/devise.rb`

```
Devise.setup do |config|
  config.mailer_sender = "please-change-me-at-config-initializers-devise@example.com"
  require 'devise/orm/active_record'
  config.case_insensitive_keys = [ :email ]
  config.strip_whitespace_keys = [ :email ]
  config.stretches = Rails.env.test? ? 1 : 10

  config.use_salt_as_remember_token = true
  config.reset_password_within = 2.hours
  config.sign_out_via = :get
End
```

app/models/devise.rb

```
class User < ActiveRecord::Base
  has_many :projects


  # Include default devise modules. Others available are:
  # :token_authenticatable, :encryptable, :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable


  # Setup accessible (or protected) attributes for your model
  attr_accessible :email, :password, :password_confirmation, :remember_me
end
```

Vygenerované views obsahovali viaceré obrazovky, najdôležitejší pre prihlasovanie bol view pre sign-in:

views/devise/sessions/new.html.erb

```
<h2>Sign in</h2>

<%= form_for(resource, :as => resource_name, :url => session_path(resource_name)) do |f| %>
  <div><%= f.label :email %><br />
  <%= f.email_field :email %></div>

  <div><%= f.label :password %><br />
  <%= f.password_field :password %></div>

```

```
10    <% if devise_mapping.rememberable? -%>
11      <div><%= f.check_box :remember_me %> <%= f.label :remember_me %></div>
12    <% end -%>
13
14    <div><%= f.submit "Sign in" %></div>
15  <% end %>
16
17  <%= render :partial => "devise/shared/links" %>
```

2.1.2 Testovanie

Testovali sme pomocou Cucumber testov, na preverenie autentifikácie boli použité nasledovné scenáre:

Kroky scenára	Akcia scenára
Logging in with non-existing account	
Given I am logged in And no user with login tkramar@gmail.com exists When I login as tkramar@gmail.com with password „password“ Then I should not be logged in And I should see that i have entered wrong account information	Po navigovaní na prihlasovaciu stránku a vyplnení údajov pre neexistujúce konto sa zobrazí upozornenie „Invalid email or password“
Logging in with existing account	
Given i am not logged in And a user with pin tkramar@gmail.com and password „secret“ exists When i login as tkramar@gmail.com with password „secret“ Then i should be logged in	Po navigovaní na prihlasovaciu stránku a vyplnení formulára s platnými údajmi sa používateľ prihlási a stránka ho presmeruje na domovskú stránku aplikácie

2.2 Registrácia používateľov

Názov user story	Popis
Registrácia používateľov	Ako používateľ sa chcem registrovať, aby som sa mohol prihlásiť a mať výhody z toho vyplývajúce

2.2.1 Implementácia

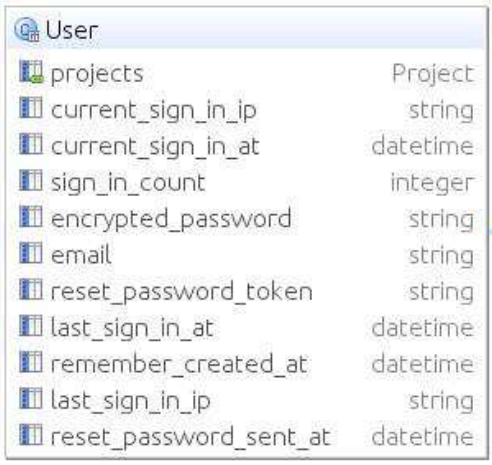
Registrácia používateľa bola implementovaná pomocou rovnakého gem-u ako autentifikácia – Devise. Bol nakonfigurovaný aby prijímal ako meno valídny e-mail a heslo s minimálne šiestimi znakmi.

Vygenerovaný bol nasledovný view:

views/devise/registrations/new.html.erb:

```
<h2>Sign up</h2>

<%= form_for(resource, :as => resource_name, :url =>
registration_path(resource_name)) do |f| %>
  <%= devise_error_messages! %>

  <div><%= f.label :email %><br />
  <%= f.email_field :email %></div>

  <div><%= f.label :password %><br />
  <%= f.password_field :password %></div>

  <div><%= f.label :password_confirmation %><br />
  <%= f.password_field :password_confirmation %></div>

  <div><%= f.submit "Sign up" %></div>
<% end %>

<%= render :partial => "devise/shared/links" %>
```

Vytváral sa tu aj model pre user-a, ktorý vyzerá následovne:



2.2.2 Testovanie

Testovali sme pomocou Cucumber testov, na preverenie registrácie boli použité nasledovné scenáre:

Kroky scénara	Akcia scenára
Registering	
Given no user with login tkramar@gmail.com exists When I register as tkramar@gmail.com and choose my password to be secret And I login as tkramar@gmail.com with password “secret” Then I should be logged in	Po navigovaní na Sign-up stránku a vyplnení formulára údajmi sa zobrazí informácia o úspešnej registrácií a automaticky sa používateľ prihlási.
Registering with already taken account name	
Given a user with login tkramar@gmail.com already exists When I register as tkramar@gmail.com and choose my password to be “secret” Then I should not be registered And I should see that the username tkramar@gmail.com is already taken	Po registrovaní s údajmi používateľa, ktorý už existuje v systéme sa zobrazí chybové upozornenie.

2.3 Vytváranie projektov

Názov user story	Popis
Vytváranie projektov	Ako používateľ si chcem vytvoriť projekt, aby som mohol logicky zhlukovať dokumenty, ktoré budem písať

Vytváranie projektov v sebe zahŕňa tieto podúlohy:

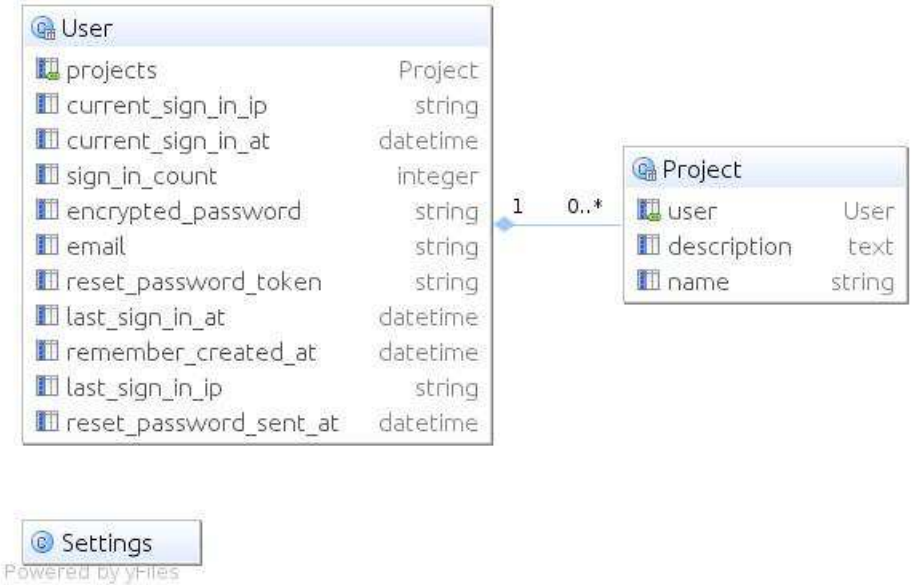
- Navrhnúť a vytvoriť tabuľky pre vytváranie projektov
- Inštalácia GRIT gemu

- Ruby interakcia s Gritom
- Cucumber testy

Táto feature je úzko prepojená s Dashboardom, kde sa zobrazujú samotné projekty.

2.3.1 Implementácia

Bola vytvorená tabuľka pre projekty (viď obrázok), v ktorej sa ukladajú potrebné dáta.



Ďalej bol nainštalovaný gem Grit, ktorý dáva objektovo-orientovaný prístup do Git repozitárov pomocou Ruby. Pri vytváraní projektu sa prostredníctvom tohto gemu vytvorí na súborovom systéme repozitár s cestou: git_path/ID_USER/ID_Project, pričom git _path je nastavený v config.yml súbore a id_user je ID aktuálne prihláseného používateľa. Celú funckionalitu má na starosti controller projects_controller.rb:

app/controlles/projects_controller:

```
1 class ProjectsController < ApplicationController
2   include ApplicationHelper
3
4   before_filter :find_project, :only => [:show, :edit, :update, :destroy]
5
6   def show
```

```
7     authorize! :read, @project
8   end
9
10  def new
11    @project = Project.new
12  end
13
14  def create
15    @project = Project.create(params[:project].merge(:owner => current_user))
16
17    if @project.save
18      flash[:notice] = t('project.created', :name => @project.name)
19      redirect_to dashboard_user_path
20    else
21      render :action => :new
22    end
23  end
24
25  def edit
26    authorize! :edit, @project
27  end
28
29  def update
30    if @project.update_attributes(params[:project])
31      flash[:notice] = t('project.updated', :name => @project.name)
```

```
32        redirect_to dashboard_user_path
33      else
34        render :action => :edit
35      end
36      authorize! :edit, @project
37    end
38
39    def destroy
40      if @project.destroy
41        flash[:notice] = t('project.deleted', :name => @project.name)
42        redirect_to dashboard_user_path
43      else
44        flash[:alert] = t('project.not_deleted', :name => @project.name)
45        redirect_to dashboard_user_path
46      end
47      authorize! :edit, @project
48    end
49
50    private
51
52    def find_project
53      @project = Project.find(params[:id])
54    end
55  end
```

Pomocou metód create, show, edit a destroy sa vytvorí nový projekt, zobrazia existujúce projekty pre aktuálne prihláseného používateľa, zmení sa názov projektu a zmaže sa celý projekt.

2.3.2 Testovanie

Pri testovaní sa predpokladá, že používateľ je prihlásený

Kroky scenára	Akcia scenára
Creating project	
When I go to my dashboard When I follow link to create new project And I create a new project called “iit.src” Then I should have a project “iit.src” among my projects	Po navigovaní do dashboard, kliknem na tlačítko “new project”, vyplním príslušné údaje a kliknem na “Create project”. V databáze sa vytvorí záznam a na dashboard sa zobrazí mnou vytvorený projekt
Editing project	
Given I have a project “iit.src” And I edit the project When I change its name to “iit.src 2012” and its description to “iit 2009” Then I should be on my dashboard And I should see that I have a project “iit.src 2012”	Na dashboarde mám vytvorený projekt “iit.src”. po kliknutí na link “edit” môžem meniť názov projektu a popis. Keď ho zmením danými údajmi a kliknem na tlačítko “Update Project”, v dashboarde vidím, že projekt je zmenený.
Delete project	
Given I have a project “iit.src” And I am on the dashboard When I delete the project Then I should have no project “iit.src”	V dashboarde mám projekt “iit.src”. Po kliknutí na tlačítko “delete” sa daný projekt vymaže a už nie je zobrazený v dashboarde.

2.4 Dashboard

Názov user story	Popis
Dashboard	Ako používateľ chcem vidieť celú pracovnú plochu na jednom mieste – dashboard, aby som mal rýchly prehľad o mojich projektoch a súboroch

2.4.1 Implementácia

Dashboard je východiskový bod po prihlásení používateľa. Sú v ňom prehľadne zobrazené všetky projekty a ich súbory, na ktorých používateľ pracuje.

Implementovaný je vo view dashboard.html.erb, využíva funkcionalitu vytvárania projektov z predošlej úlohy.

app/views/users/dashboard.html.erb:

```
1  <h3>Dashboard</h3>
2
3  <table title="dashboard" border="1" cellpadding="3" cellspacing="0" id="dashboard">
4    <tr>
5      <th>Name</th>
6      <th>Owner</th>
7      <th>Last modified</th>
8      <th></th>
9      <th></th>
10   </tr>
11   <% @projects.each do |project| %>
12     <tr>
13       <td><%=link_to project.name, project_path(project)%></td>
14       <td><%= project.owner.email %></td>
15       <td><%= time_ago_in_words(project.updated_at)%> ago</td>
```

```
16        <td><%= link_to 'Edit', edit_project_path(project) %></td>
17        <td><%= link_to 'Delete', project, :confirm => 'Are you sure you want to delete
this project?', :method => :delete %></td>
18      </tr>
19      <% end %>
20      </tr>
21  </table>
22
23  <br>
24  <%= link_to 'New Project', new_project_path %>
```

2.4.2 Testovanie

Kroky scenára	Akcia scenára
Seeing dashboard after login	
Given I am logged out When I log in Then I should be on my dashboard page	Po prihlásení sa zobrazí dashboard.
Seeing list of projects and their files in dashboard	
Given I have the following projects and files: Project: tpcup Files: main.tex, overview.jpg Project: iitsrc Files: iitsrc.tex, bibliography.bib When I go to my dashboard Then I shuld see that I have project tpcup And I shoyld see that project tpcup has files main.tex, overview.jpg Then I should see that I have project iit.src	V dasbhoarde mám zoznam mojich projektov spolu so súbormi, ktoré k nim patria.

Seeing last change for a project and each file	
Given I am logged in Given I have project iitsrc And the project has file main.tex which was last modified 1 days ago And the project has file bibliography.bib which was last modified 3 days ago When I go to my dashboard Then I should see that the project iitsrc was last modified 1 day ago And I should see that the file main.tex was last modified 1 day ago And I should see that the file bibliography.bib was last modified 3 days ago	V dashboarde je zoznam projektov so súbormi a údaji o poslednej zmene súboru/projektu.

2.5 Autorizácia

Názov user story	Popis
Autorizácia	Ako používateľ by som mal byť jediný, ktorý vidí moje súbory, aby som obmedzil prístup k projektom, ktoré vytváram.

V rámci tejto úlohy boli riešené nasledovné podúlohy:

- Nainštalovať a naštudovať CanCan
- Definovať CanCan ability
- Zavolanie CanCan pred zobrazením projektu
- Cucumber testy

2.5.1 Implementácia

Pri autorizácií sa využíva gem CanCan, ktorý obmedzuje prístup používateľov k prostriedkom. Zabezpečujeme aby používateľ nemohol zmeniť URL a touto zmenou by získal prístup k inému projektu. Všetky prístupy sú riešené cez jednu Ability triedu. Táto trieda je nastavená tak, aby zobrazovala používateľom iba tie projekty, ktorých sú vlastníkom. V application_conrtoller.rb musí

byť definovaná akcia, ktorá sa vykoná v prípade zamietnutia CanCanom, riešime to pomocou presmerovania na public stránku forbieden 403. Ďalej v každom cntrolleri musí byť v príslušnej metóde povedané, že má byť volaný CacCan. To sa zabezpečí napríklad volaním:

```
authorize! :edit, @project
```

Kód hlavnej triedy CanCanu je definovaný nižšie, tento kód však bude nutné pri vývoji produktu prispôsobovať požiadavkám.

app/models/ability.rb:

```
1  class Ability
2    include CanCan::Ability
3
4    def initialize(user)
5      user ||= User.new
6      can [:read, :edit], Project do |project|
7        project.owner == user
8      end
9    end
10 end
```

2.5.2 Testovanie

Testovanie sme vykonávali pomocou Cucumber testov.

Kroky scenára	Akcia scenára
Only author can see his projects	
Given I am logged in as tkramar@gmail.com And I have a project “iit.src” And I go to my dashboard Then I should be able to see the project “iit.src” When I relogin as mbielik@gmail.com And I go to my dashboard	Pri prihlásení sa dashboard zobrazuje iba projekty aktuálne prihláseného používateľa.

Then I should not see the project “iit.src”	
Projects are visible to their authors	
Given I am logged in as tkramar@gmail.com And I have a project “iit.src” And there is another user and she has a project “papers” When I go to the “iit.src” project page Then I should be able to see the project When I go to the “papers” project page Then I should not be able to see the project	Pri manuálnom zadaní projektu do linku, na ktorý nemá aktuálny používatel právo, sa zobrazí error 403 – forbidden.

2.6 Úprava súborov

Ako používateľ, chcem mať možnosť upravovať súbory, aby som mohol vytvárať dokumenty na webe, bez nutnosti inštalovať TEX na mojom počítači.

V čase písania dokumentácie ešte táto úloha nebola vyriešená z dôvodu personálnych problémov v tíme, presun do ďalšieho šprintu.

2.7 Verziovanie súborov

Táto funkcionalita bola presunutá do ďalšieho šprintu z dôvodu priamej závislosti na vytváraných súboroch. Vytváranie súborov bolo tiež riešené v ďalšom šprinte.

2.8 Dizajn a user experience

Ako prevádzkovateľ editora chcem, aby editor dobre vyzeral a pohodlne sa používal, aby ľudia za neho boli ochotní platiť.

V čase písania dokumentácie ešte táto úloha nebola vyriešená z dôvodu personálnych problémov v tíme, presun do ďalšieho šprintu.

3 Tretí šprint - Cecília

3.1 Úprava súborov

Názov user story	Popis
Úprava súborov	Ako používateľ chcem mať možnosť upravovať súbory, aby som mohol vytvárať dokumenty na webe pohodlne, bez nutnosti inštalovať Tex na mojom počítači.

Úprava súborov je jedna z ťažísk aplikácie WebTex. V tomto user story bolo treba implementovať funkcionalitu vytvárania, mazania a úpravy súboru a dbať a spolupracovať s feature Verziovania súborov.

- Cucumber TEST
- Vytvorenie súboru
- Editovanie mena súboru
- Mazanie súboru
- Editovanie obsahu súboru
- Ukladanie a verziovanie

3.1.1 Implementácia

Naša aplikácia je riešená tak že každému používateľovi prislúcha v súborovom systéme jedna zložka nazvaná jeho ID číslom. V zložke má projekty ktorých je vlastníkom, ktoré sú pomenované ID číslom projektu. V každej zložke projektu môžeme mať rôzne súbory.
Ťažisko funkcionality je v dvoch súboroch:

- **app/controllers/documents_controller.rb**
- **app/models/documents.rb**

Zdrojový kód týchto súborov je umiestnený v kapitole 3.2 Verziovanie súborov. Obe funkcionality boli ako Úprava súborov a Verziovanie dokumentov boli implementované do týchto dvoch súborov a boli implementované v spolupráci oboch zodpovedných programátorov.

3.1.2 Testovanie

Testovanie sme vykonávali pomocou Cucumber testov.

Kroky scenára	Akcia scenára
BACKROUND	
Given I am logged in And I have project "iit.src" And The project "iit.src" has file "main.tex" with content """ This is content for file """	Existuje projekt „it.src“ so súborom „main.text“.
Scenario: Creating new file	
When I go to project detail And I create a new file called "overview.jpg" Then I should have a file "overview.jpg" among my files	Keď vytvorím súbor s názvom „overview.jpg“ tak ho neskôr budem vidieť v projekte.
Scenario: Opening the file	
When I open the file "main.tex" Then I should see that the file "main.tex" has content """ This is content for file """	Keď otvorím súbor, budem vidieť jeho obsah.
Scenario: Editing the file content	
When I open the file "main.tex" And I change the content of file "main.tex" to """ Edited text. """ And I open the file "main.tex" Then I should see that the file "main.tex" has content """ Edited text.	Keď do súboru „main.tex“ vpíšem pri editácii text, budem vidieť tento tex aj keď súbor znovu otvorím.

"""	
Scenario: Delete file	
When I delete the file "main.tex" Then I should have no file "main.tex"	Keď zamažem súbor, potom ho už nebudem vidieť medzi mojimi súbormi.
Scenario: Edit file name	
When I rename the file "main.tex" to "iitsrc.tex" Then I should have no file "main.tex" But I should have file "iitsrc.tex"	Keď premenujem súbor, tak sa zmení jeho meno.

3.2 Verziovanie súborov

Názov user story	Popis
Verziovanie súborov	Ako používateľ chcem vidieť verzie dokumentu, aby som videl ako sa dokument vyvíjal v čase

V rámci úlohy sme riešili nasledovné podúlohy:

- Zobrazenie zoznamu revízií
- Zobrazenie konkrétnej revízie
- Cucumber testy

Verziovanie dokumentov projektu je jeho dôležitá súčasť v kontexte vývoja dokumentu čase. Jednotlivé verzie dokumentu sa vytvoria v prípade, že používateľ, ktorý je na tomto dokumente oprávnený pracovať, upraví tento dokument a uloží ho. Úlohou verziovania je pre daný projekt zobraziť všetky k nemu prislúchajúce verzie a takisto zobraziť aj konkrétnu verziu dokumentu. Túto záležitosť sme sa rozhodli riešiť pomocou verziovacieho nástroja Git, konkrétne v projekte WebTex bude využitý gem Grit.

3.2.1 Implementácia

Naša aplikácia je riešená tak že každému používateľovi prislúcha v súborovom systéme jedna zložka nazvaná jeho ID číslom. V zložke má projekty ktorých je vlastníkom, ktoré sú pomenované

ID číslom projektu. V každej zložke projektu môžeme mať rôzne súbory. Revízie týchto súborov sú uložené v gite takže žiadne fyzické kópie na disku. Práve týmito revíziami sa zaoberá táto implementácia. Hlavné ťažisko funkcionality je documents_controller.rb a v modeli documents.rb

app/controllers/documents_controller.rb:

```
1  class DocumentsController < ApplicationController
2    before_filter :find_document
3
4
5    def revisions
6
7    End
8
9    def show
10
11   End
12
13   def revision
14     @document = @project.find_file_with_sha(params[:file_name], params[:revision_id])
15     @document.at_revision(params[:revision_id])
16   end
17
18   def save
19     @document.save(params[:content])
20     redirect_to show_document_path(params[:project_id], params[:project_name],
   params[:file_name])
21   end
22
```

```
23   def destroy
24     @document.delete
25     redirect_to project_path(params[:project_id])
26   end
27
28   def update
29     @document.update(params[:new_file_name])
30     redirect_to show_document_path(params[:project_id], params[:project_name],
params[:new_file_name])
31   end
32
33   def find_document
34     @project = Project.find(params[:project_id])
35     @document = @project.find_file_with_sha(params[:file_name], params[:revision_id])
36   end
37 End
```

Práca s dátami je realizovaná pomocou modelu pre dokumenty.

app/models/documents.rb:

```
1 class Document
2
3   def initialize(options)
4       if(options.length == 3)
5         @file_name = options[:file_name]
6         @repository = options[:repository]
7         @file_path = options[:git_project_path]
```

```
8       else
9         @file_name = options[:file_name]
10        @repository = options[:repository]
11        @sha = options[:sha]
12        @file_path = options[:git_project_path]
13      end
14    end
15    attr_reader :file_name
16
17    def at_revision(sha)
18      Document.new :file_name => @file_name, :repository => @repository, :sha => sha,
:git_project_path => @file_path
19    end
20
21    def revision_content
22      tree = @repository.tree(@sha)
23      file = tree/"#{@file_name}"
24      file.data
25    end
26
27    def revisions
28      @repository.log("master", @file_name).reverse
29    end
30
31    def name
32      return @file_name
```

```ruby
  end

  def content
    file_path_with_name = File.join(@file_path, @file_name)
    file = File.open(file_path_with_name, "r")
    return file.read
  end

  def save(contents)
    commit(@file_name, "Add file #{@file_name}") do |file_path|
      File.open(file_path, "w") do |f|
        f << contents
      end
    end
  end

  def update(new_file_name)
    old_file_path = File.join(@file_path, @file_name)

    commit(new_file_name, "Rename file #{@file_name} to #{new_file_name}") do
|new_file_name|
      File.rename(old_file_path, new_file_name)
    end
  end

  def delete
```

```
58      commit(@file_name, "Delete file #{@file_name}") do |file_path|
59        File.delete(file_path)
60      end
61    end
62
63    def last_modified_date
64      @last_modified_date ||= Time.now - @repository.log("master", @file_name,
:max_count => 1).last.committed_date
65    end
66
67    private
68    def commit(file, message)
69      file_path = File.join(@file_path, file)
70
71      yield(file_path)
72
73      Dir.chdir(@file_path) do
74        @repository.add file
75        @repository.commit_all "#{message}"
76      end
77    end
78  end
```

Popis najdôležitejších častí kódu:

```
class DocumentsController < ApplicationController
  before_filter :find_document
  #toto vytvori instanciu Document s nazvom @document pomocou metody modelu
Project
  #find_file_with_sha(params[:file_name], params[:revision_id])
```

```
  def revisions
    #tuto je prepojenie view document/revisions.html.erb s modelom, ktory posle
premennu @document. to je instancia
    #modelu Document.ako mozes vidiet v tom viewe sa potom deje cely vypis
zoznamu revizii s tym kto commitoval, kedy...
  end

  def show

  end

  def revision
    @document = @project.find_file_with_sha(params[:file_name],
params[:revision_id])
    @document.at_revision(params[:revision_id])
    #toto komunikuje s view document/revision.html.erb, at_revision je fcia
modelu , ktora nastavi aktualnu sha instancii @document
    #a obsah suboru sa vypise @document.revision_content
  end
```

Revízie súborov sú závislé na modeli projektov, ktorý pacuje s dátami a pri vytváraní projektu vytvára príslušný repozitár. Contoller revízii teda musí byť dobre prepojený s týmto modelom. Potom už je potrebný iba view na zobrazovanie týchto revízii v prehliadači.

3.2.2 Testovanie

Testovanie sme vykonávali pomocou Cucumber testov.

Kroky scenára	Akcia scenára
BACKROUND	
Given I am logged in as "tkramar@gmail.com" And I have project "iit.src" And The project "iit.src" has file "main.tex" with content """ This is the first version of the document """ And I open the file "main.tex" And I change the content of file "main.tex" to """ This is the second version of the document """	Zmením 3 krát dokukemnt „main.tex“

And I change the content of file "main.tex" to """ This is the final version of the document """ """ This is the final version of the document	
Scenario: Browsing document revisions	
When I go to "main.tex" revisions page Then I should see that "main.tex" has 3 revisions	Keď pôjdem na stránku revízii súboru uvidím 3 revízie
Scenario: Viewing document's older revision	
When I go to "main.tex" revisions page And I open the first revision of file "main.tex" Then I should see that the contents of that file at that revision were """ This is the first version of the document """	Keď som na revíziách súboru „main.tex“ a kliknem na prvú revíziu obrazí sa mi dokument v tom čase, čiže jeho prvá verzia.

3.3 Správa kolaborantov

Názov user story	Popis
Správa kolaborantov	Ako používateľ chcem spravovať kolaborantov projektu, ktorí k nemu budú mať prístup, aby som na projekte mohol pracovať s inými používateľmi.

V rámci úlohy sme riešili nasledovné podúlohy:

- Cucumber TEST
- Rozposielanie emailov kolaborantom
- Vytvorenie prepojovacej tabuľky medzi users a projects
- Úprava DASHBOARDU
- Pridanie kolaborantov
- Výpis kolaborantov
- Mazanie kolaborantov

Správa kolaborantov a vôbec kolaborácia samotná je jedným zo základných pilierov našej aplikácie WebTex. Správa kolaborantov by mala umožňovať pridávať jednotlivých spolupracovníkov ku konkrétnym projektom. Samozrejme by malo byť umožnené kolaboranta aj zmazať. Táto funkcionalita v sebe nezahŕňa len prácu s nejakou tabuľku, je potrebné aj autorizovať prístup ku k jednotlivým projektom pomocou CanCan. Dôležité je aby pridávanie emailov bolo jednoduché a rýchle. Vyriešili sme to pomocou zoznamu emailových adries. Používateľ jednoducho zadá emailové adresy používateľov, ktorých chce zadať medzi spolupracovníkov na projekte.

Samozrejme tu sa hneď vynára niekoľko špecifík pri implementácii. Správanie používateľa totiž treba ošetriť. Ak by náhodou zadal zlú adresu, aby sa mu zobrazila chybová správa. Kolaborant by nemal mať možnosť odstrániť vlastníka projektu. Taktiež by nemalo byť možné pridať 2x toho istého kolaboranta k tomu istému projektu. Je to teda množstvo ošetrení ktoré treba zohľadniť.

Významným faktom pri pridávaní kolaborantov, je aj to aby boli informovaný o tom že boli pridaný. Na to využívame gem Mailer, spolu s definovaným telom emailovej správy kde sú vložené informácie ako email kolaboranta a projekt na ktorý bol pridaný ako kolaborant. Bolo treba nakonfigurovať aj smtp. Zatiaľ využívame na rozosielanie emailov účet Gmail s názvom webtexmailer@gmail.com. Keď bude projekt umiestnený na vlastnú doménu, nie je problematické zmeniť konfiguračný súbor.

3.3.1 Implementácia

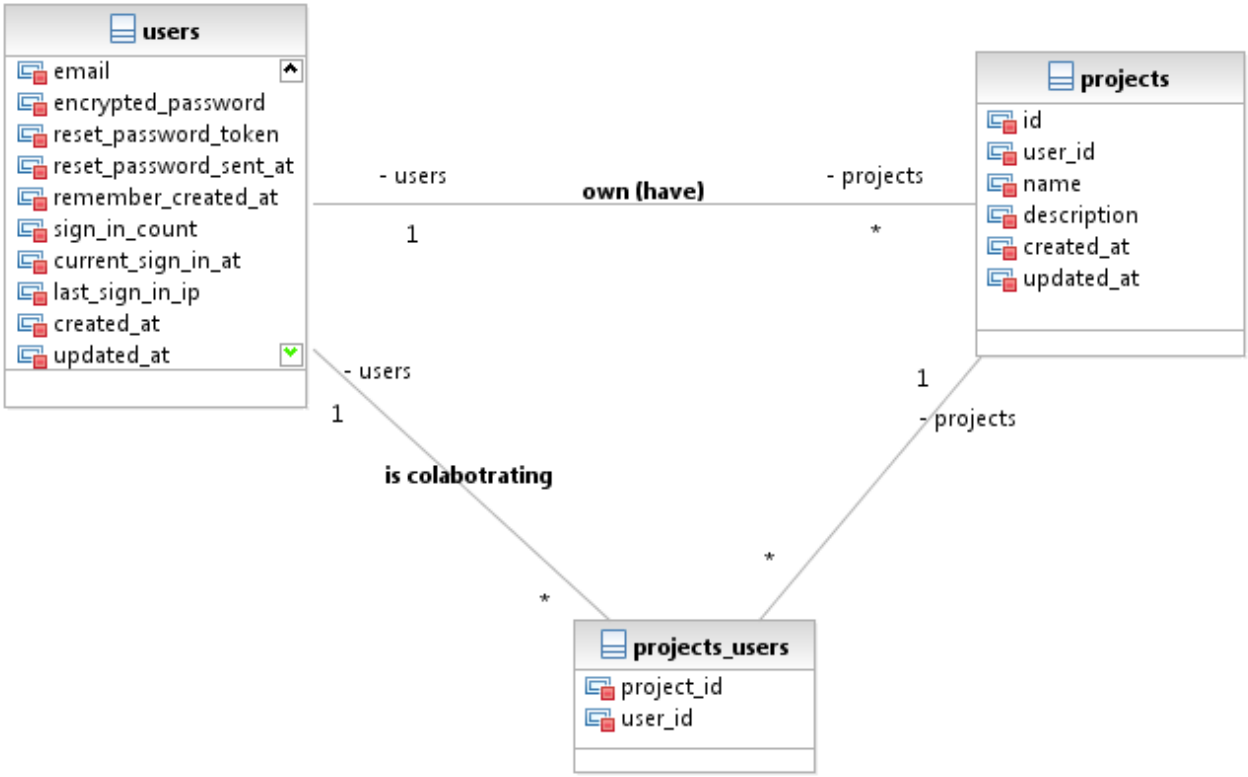
Implementácia pozostávala z niekoľkých častí. Najskôr bolo treba vytvoriť prepojovaciu tabuľku medzi používateľmi a projektmi. Toto bolo zabezpečené novou migráciou.

db/migrate/20111119220709_projects_users.rb

```
1 class ProjectsUsers < ActiveRecord::Migration
2   def change
3     create_table :projects_users, :id => false do |t|
4       t.integer :project_id, :null => false
5       t.integer :user_id, :null => false
6     end
```

```
7
8     add_index :projects_users, [:project_id, :user_id], :unique => true, :name =>
'by_project_and_user'
9   end
10 end
```

Index je tabuľke pridelený na riadok práve preto aby používateľ nemohol dvakrát kolaborovať na tom istom projekte, to by nedávalo zmysel.



Obrázok 1 – súčasný logický dátový model projektu WebTex

Dátový model sa možno zdá zatiaľ jednoduchý, ale množstvo záležitostí je riešené Gitom a samotné projekty a súbory používateľov sú uložené v súborovom systéme či už lokálneho stroja alebo pri reálnom nasadení aplikácie servera.

Ďalej bolo treba v implementácii nainštalovať gem Mailer a nastaviť mailového klienta. Nastavili sme gmailový účet. Túto konfiguráciu bolo treba realizovať súbore **/config/enviroments/development.rb** . Tu bolo potrebné pridať nasledovné riadky:

```
17  config.action_mailer.raise_delivery_errors = true

19  config.action_mailer.delivery_method = :smtp

20  config.action_mailer.smtp_settings = {

21  :address              => "smtp.gmail.com",

22  :port                 => 587,

23  :domain               => 'gmail.com',

24  :user_name            => 'webtexmailer',

25  :password             => 'aaabbb111',

26  :authentication       => 'plain',

27  :enable_starttls_auto => true  }
```

Po pridávaní kolaboranta do prepojovacej tabuľky sa volá funkcia zo súboru **app/mailers/user_mailer.rb** , tu sá nasetujú premenné, ktoré budú vystupovať v uvítacom maile.

```
1 class UserMailer < ActionMailer::Base

2   default from: "webtexmailer@gmail.com"

3

4   def welcome_email(user, project)

5     @user = user

6     @project = project

7     mail(:to => user.email, :subject => "You was add as collaborator")

8   end

9 end
```

Šablóna pre email je vlastne naštýlovaním správy do HTML. Je umiestnená v súbore: **App/views/user_mailer/wellcome_email.html.erb**:

```
1 <!DOCTYPE html>
```

```
<html>
  <head>
    <meta content="text/html; charset=UTF-8" http-equiv="Content-Type" />
  </head>
  <body>
    <h2>Hello <%= @user.email %> you were added as collaborator for project named "<%= @project.name %>"</h2>
    <p>Thanks for joining and have a great day!</p>
  </body>
</html>
```

Hlavné ťažisko vytvorenia kolaboranta je umiestnené v modeli project:

app/models/project.rb:

```
  def add_collaborators_by_email(emails)
    all_emails = emails.gsub(/\s+/, "").split(",")
    registered_users = User.where(:email => all_emails)
    registered_users.each do |user|
      if user
        users << user
        UserMailer.welcome_email(user, self).deliver
      end
    end
    valid_emails = registered_users.collect(&:email)
    invalid_emails = all_emails - valid_emails

    errors.add(:base, :email_not_found, :emails => invalid_emails.join(", "), :count => invalid_emails.length) if invalid_emails.any?
```

```
39
40      [valid_emails, invalid_emails]
41    end
```

Ďalšia funkcionalita je prenesená do controllera.

app/controllers/collaborators_controller.rb

```
1  class CollaboratorsController < ApplicationController
2     include ApplicationHelper
3
4     before_filter :find_project, :find_collaborators
5
6    def create
7     authorize! :edit, @project
8
9     valid_emails, invalid_emails =
@project.add_collaborators_by_email(params[:collaborators])
10
11    flash[:notice] = t('collaborator.created', :emails => valid_emails.join(", "),
:project => @project.name, :count => valid_emails.length) if valid_emails.any?
12
13    if invalid_emails.any?
14      @active_tab = :collaborators
15      render 'projects/edit'
16    else
17      redirect_to dashboard_user_path
18    end
19   end
```

```
20
21  def edit
22    authorize! :edit, @project
23  end
24
25  def destroy
26    @user=User.find(params[:format])
27    if @user != @project.owner
28      if @user
29        @project.users.delete(@user)
30        flash[:notice] = t('collaborator.deleted', :name => @user.email)
31        if @user == current_user
32          redirect_to dashboard_user_path and return
33        end
34      else
35        flash[:alert] = t('collaborator.not_deleted', :name => @user.email)
36      end
37    else
38      flash[:alert] = t('collaborator.not_deleted_owner', :name => @user.email)
39    end
40    @active_tab = :collaborators
41    render 'projects/edit'
42  end
43
44  private
```

```
45
46    def find_collaborators
47      @collaborators = Project.find(params[:project_id]).users
48    end
49
50    def find_project
51      @project = Project.find(params[:project_id])
52    end
53  end
```

Chybové správy sú umiestnené v súbore **config/locales/en.yml**.

K celej tejto funkcionalite je vytvorený príslušný view (**app/views/collaborators/_edit.html.erb**), ktorý zbiera parametre a posiela ich do príslušného modelu a controllera.

3.3.2 Testovanie

Testovanie sme vykonávali pomocou Cucumber testov, ktoré sme postavili rozsiahle.

Kroky scenára	Akcia scenára
BACKROUND	
Given the following users exists: "webtex1test@gmail.com", "webtex2test@gmail.com", "webtex3test@gmail.com", "webtex4test@gmail.com" Given I login as "webtex1test@gmail.com" And I have a project "iitsrc" And I want to edit collaborators on project "iitsrc"	Existujú vymenovaní používatelia. Som prihlásený ako používateľ webtex1test@gmail.com a mám projekt „itsrc“. A idem editovať kolaborantov k tomuto projektu.
Scenario: Collaborator can see project on dashboard	
And I go to collaborators tab And I add new collaborators "webtex2test@gmail.com,	Keď pridám nových kolaborantov, tak budú po

webtex3test@gmail.com" When I login as "webtex2test@gmail.com" Then I should see that I have a project "iitsrc" When I login as "webtex3test@gmail.com" Then I should see that I have a project "iitsrc"	svojom prihlásení vidieť daný projekt na svojom dashboarde.
Scenario: User which is NOT between collaborators can not see project on dashboard	
And I go to collaborators tab And I add new collaborators "webtex2test@gmail.com, webtex3test@gmail.com" When I login as "webtex4test@gmail.com" Then I should not see the project "iitsrc"	Keď používateľ nie je v zozname kolaborantov projektu, nemôže vidieť projekt na svojom dashboarde.
Scenario: Collaborator can delete other collaborator	
And I go to collaborators tab And I add new collaborators "webtex2test@gmail.com, webtex3test@gmail.com" When I delete collaborator "webtex2test@gmail.com" of project "iitsrc" And I login as "webtex2test@gmail.com" Then I should not see the project "iitsrc"	Kolaborant môže z tabuľky kolaborantov zmazávať iných kolaborantov.
Scenario: Collaborator can not delete owner	
And I go to collaborators tab And I add new collaborators "webtex2test@gmail.com, webtex3test@gmail.com" When I login as "webtex2test@gmail.com" And I want to edit collaborators on project "iitsrc" And I delete collaborator "webtex1test@gmail.com" of project "iitsrc" Then I should still see owner "webtex1test@gmail.com" in table of collaborator for project "iitsrc" When I login as "webtex1test@gmail.com" Then I should see that I have a project "iitsrc"	Kolaborant nemôže zmazať vlastníka projektu zo zoznamu kolaborantov.
Scenario: User which is NOT between collaborators can not see collaborators, when change url	

And I add new collaborators "webtex2test@gmail.com, webtex3test@gmail.com" When I login as "webtex4test@gmail.com" And I want to edit collaborators on project "iitsrc" Then I should not be able to see collaborators	Tu je testovaná autorizácia, aby nemohol pristupovať k zmene kolaborantov nekolaborant.
Scenario: If bad email is input, will be not added to table of collaborators	
And I go to collaborators tab When I add new collaborators "blbost-webtech@gmil.com" Then I should not see collaborators "blbost-webtech@gmil.com" in table of collaborators for project "iitsrc"	Nie je možné pridať používateľov, ktorí nie sú registrovaný s platným emailom.
Scenario: Sending emails with message, that user was added as collaborator for project	
And I go to collaborators tab When I add new collaborators "webtex4test@gmail.com" Then "webtex4test@gmail.com" should receive an email	Kolaborantovi ktorí bol pridaní k spolupráci na projekte príde informačný email.

3.4 Dizajn a UX

Názov user story	Popis
Dizajn a UX	Ako používateľ chcem aby rozhranie bolo pre mňa zaujímavé vizuálne príťažlivé a zároveň prehľadné

Kvalitný návrh a grafický dizajn bývajú tým prvým prvkom, ktorým si aplikácia môže získať používateľa. Najskôr musí byť rozhranie teda príťažlivé a potom sa používateľ odhodlá vyskúšať funkcionalitu. Keďže ide o webovú aplikáciu je vhodné použiť technológie ako CSS3, compass-style, SCSS, HTML5, JAVAscript.

V rámci úlohy boli riešené nasledovné podúlohy:

- Návrh obrazoviek
- Štúdium CSS, SCSS, compass, HTML5
- Naštýlovanie zatiaľ implementovaných obrazoviek (login, dashboard, úprava projektov, editácia kolaborátorov, editácia súborov a revízie)

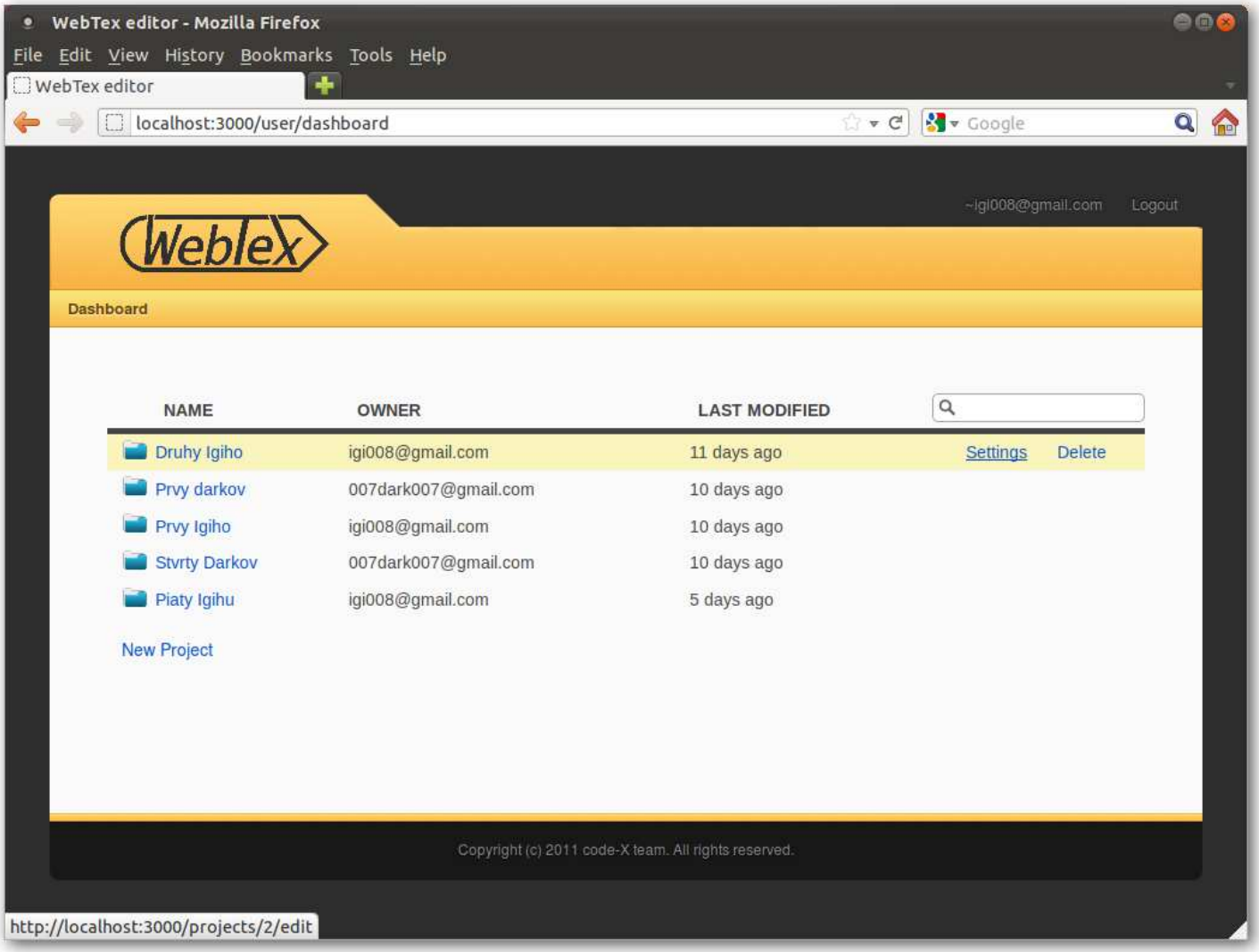
Na koľko ide o rozsiahlu a pomerne náročnú úlohu, jej realizácia prebieha naprieč viacerými šprintmi.

Pre realizácii rozhraní používame Compass ide o open-source framework pre CSS. Mnohé veci sú tu riešenie prehľadnejšie a pre programátora sa to používa lepšie lebo sa nemusí príliš opakovať. Pre dynamické prvky bol použitý najmä Javascript a jQuery.

Celý dizajn je zameraný na súlad farieb vizuálnu príťažlivosť, jednoduchosť a intuitívnosť. Logo bolo navrhnuté v programe Adobe Illustrator CS5 pomocou vektorovej grafiky, čo tiež značí o profesionálnom prístupe k dizajnu.

Každý jeden view musel byť naštýlovaný do finálnej podoby. Potrebný kód teda môžeme vidieť v jednotlivých viewoch.

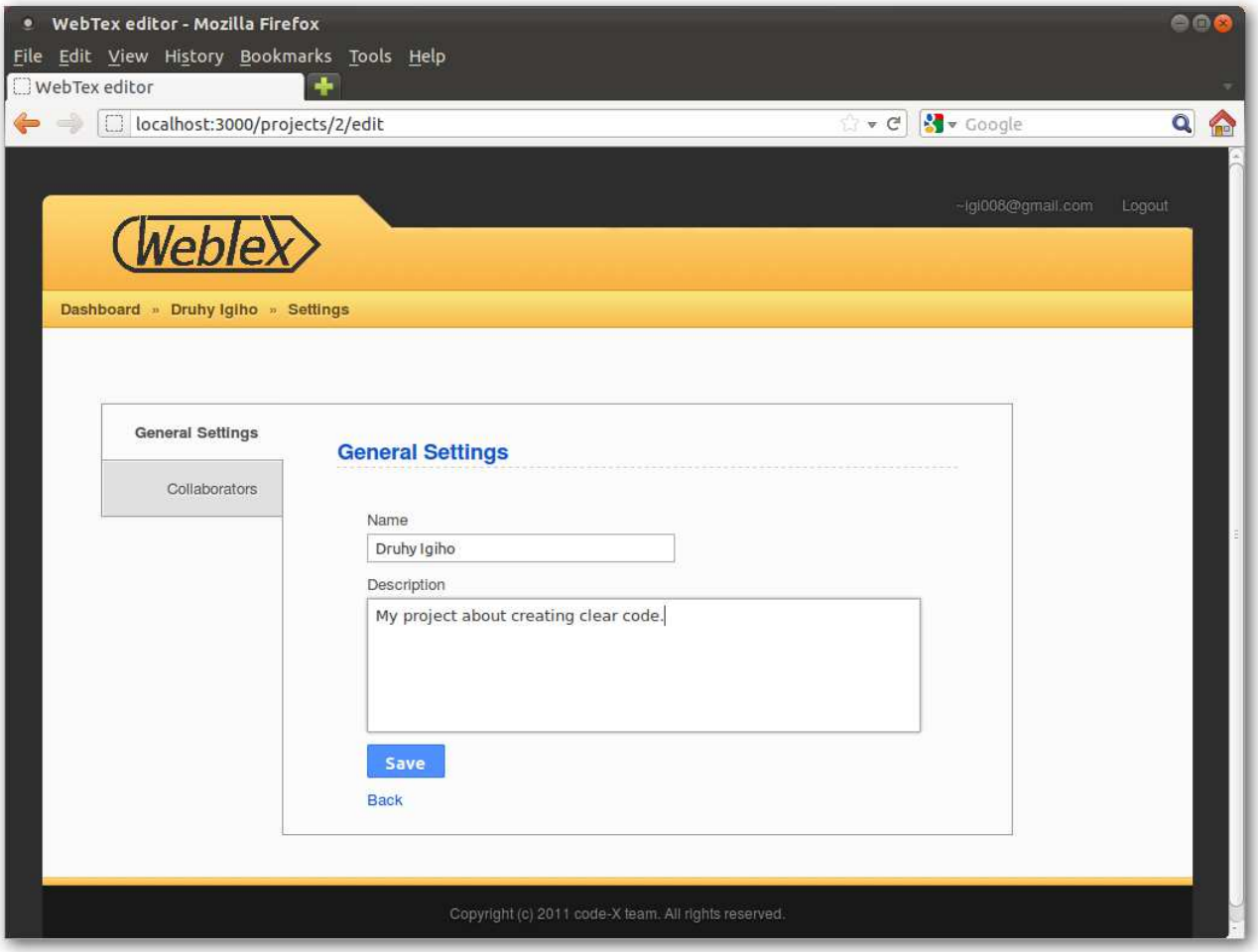
Tu ponúkame ukážku ako vyzerajú niektoré obrazovky nášho editora WebTex.



Obrázok 2 – Obrazovka Dashboardu editora Webtex

Na dashboarde používateľ môže vidieť svoje projekty a projekty na ktorých kolaboruje, to jednoducho zistí pomocou položky owner. Pri pohybe kurzora po dashborde sa zvýrazňuje konkrétny projekt a objavujú sa položky napravo nazvané Setting a Delete. Delete slúži na

jednoduché zmazanie projektu, pričom môžem zmazať iba svoj projekt, nie cudzí. Významná je však položka Settings, ktorá nám sprístupní nastavenie projektu.



Obrázok 3 – Obrazovka nastavení projektu editora Webtex

V nastaveniach projektu je nám umožnené zmeniť meno a opis projektu. Na karte Collaborator nám je umožnené plnohodnotne pridávať a odstraňovať spolupracovníkov projektu.

Na Dashboarde máme možnosť kliknúť na názov projektu, ktorý nás odnaviguje práve k súborom tohto projektu, kde ich môžeme jednoducho editovať. Môžeme meniť ich meno, ale aj obsah. Tu už pracuje aj verziovanie ktoré je založené n nástroji Git. Potom si môžem pozrieť históriu vývoja súboru pri kliknutí na Revisions, ktoré je dostupné pri každom súbore.

4 Štvrtý šprint - Dagmar

Tu sme si definovali úlohy hlavne charakteru štýlovania a prezentácie tímu ako aj tvorbu dokumentácie. Nezabudli sme ani na dôležitú funkciu a začali sme pracovať na Editore a Kompilácii kódu.

Úloha	Plnenie
Naštýlovať kolaborantov	Upraviť view kolaborantov do takej podoby, aby bol v súlade so štýlom projektu.
Naštýlovať strom súborov a zobraziť ho	Toto si vyžaduje implementáciu, aby sme mohli súbory zobrazovať v stromovej štruktúre, preto bude tomuto venovaná samostatná **kapitola (4.1)**.
Naštýlovať revízie súborov	Upraviť view revízii do takej podoby, aby bol v súlade so štýlom projektu.
Editor súborov	Editor súborov je dôležitým elementom najmä koli prehľadnosti pri editovaní, číslovanie riadkov, zvýrazňovanie syntaxe a iné. Budeme transformovať riešenie CODEMIROR pre naše účely. Vyžaduje si implementáciu, a preto sa tomu bližšie venujeme v samostatnej **kapitole (4.2)**.
Kompilácia kódu	Významný prvok aplikácie, v tomto šprinte sa venujeme iba čistému prekladu súboru. Vyžaduje si implementáciu, preto je bližšie popísaná v samostatnej **kapitole (4.3)**.
Nová webová stránka tímu	Nová webová stránka bola implementovaná za použitia CSS, HTML a JAVAscriptu. Stará webová stránka nebola najšťastnejšia a programátor, ktorý ju programoval náš tím opustil. Preto sme sa rozhodli vytvoriť novú, zaujímavejšiu a prehľadnejšiu.
Preinštalovať server	Server je nutné preinštalovať nastali určité problémy pri inštalácii Capistrano. Prechádzame na Fedoru a inštalujeme všetky potrebné veci, ktoré budú nutné na nasadenie našej aplikácie.
Dokumentácia k projektu	Ide o zdokumentovanie implementácie ktorá prebiehala

	v jednotlivých šprintoch. Niektoré šprinty boli dlhšie ako 2 týždne, ale bolo to najmä kôli odchodom členov tímu a rozsah funkcionality implementovaný v týchto šprintoch bol dosť široký.
Prezentácia projektu	Príprava materiálov na prezentáciu prototypu projektu.

4.1 Zobraziť strom súborov a naštýlovať ho

Názov user story	Popis
Zobraziť strom súborov a naštýlovať ho	Ako používateľ chcem aby som mal možnosť vidieť súbory v stromovej štruktúre

Realizácia nie je v čase odovzdávania dokumentácia dokončená. Úloha je v PROGRESE.

4.2 Editor súborov

Názov user story	Popis
Editor súborov	Ako používateľ chcem aby som mal možnosť editovať súbory v prehľadnom editore s číslovaním riadkov a zvýrazňovaním syntaxe

Realizácia nie je v čase odovzdávania dokumentácia dokončená. Úloha je v PROGRESE.

4.3 Kompilácia kódu

Názov user story	Popis
Kompilácia kódu	Ako používateľ chcem aby som mal možnosť súbory LaTexu kompilovať do PDF.

Úlohou kompilácie je prekompilovať zadaný kód napísaný v Latexu do súboru formátu „pdf“ a uložiť tento súbor do adresára .../public/pdf/
Na kompiláciu kódu sa využíva nástroj Texlive.

Tu sú najdôležitejšie riadky kódu:

```
out = "#{Rails.root}/public/pdf/"
system("pdflatex -output-directory #{out} #{@document.file_path}")
redirect_to show_document_path(params[:project_id],
params[:project_name], params[:file_name])
```

V kóde je cesta k cieľovému adresáru zapísaná v premennej out. Následne sa využije príkaz Ruby systém, ktorému ako parameter vložíme príkaz Shellu pdflatex. Tento príkaz má prepínač -output-directory, to znamená, že musíme zadať aj priečinok kde sa pdf súbor uloží, to je v prennej out. Druhý parameter je cesta k súboru, ktorý obsahuje kód latexu. Tento súbor nemusí mať príponu tex alebo iné prípony špecifikujúce latexovské súbory.

5 Prototyp projektu WebTex

Náš prototyp je v podstate určitým naplnením Backlogu projektu. Je implementovaná značná časť funkcionality. Najvýznamnejšie jadro aplikácie je plne funkčné a dokonca je prototyp použiteľný.

#	Project	Tracker	Status	Priority	Subject	Assignee	Category	Target version	Start date	Due date	% Done
Backlog (40)											
1036	WebTex editor	Feature	In progress	Normal	Neprihlásený používateľ		Backlog	Backlog	13.10.2011	11.12.2011	
1029	WebTex editor	Feature	New	Normal	Zobrazenie „about" stránky		Backlog	Backlog	13.10.2011		
1030	WebTex editor	Feature	Closed	Normal	Prihlásenie používateľa		Backlog	Backlog	13.10.2011	09.11.2011	
1031	WebTex editor	Feature	Closed	Normal	Registrácia používateľa		Backlog	Backlog	13.10.2011	09.11.2011	
1041	WebTex editor	Feature	Closed	Normal	Možnosť prekladu LaTeX kódu		Backlog	Backlog	13.10.2011	11.12.2011	
1037	WebTex editor	Feature	In progress	Normal	Prihlásený používateľ		Backlog	Backlog	13.10.2011	30.11.2011	
1035	WebTex editor	Feature	Closed	Normal	Možnosť vytvárania nových projektov		Backlog	Backlog	13.10.2011	14.11.2011	
1034	WebTex editor	Feature	Closed	Normal	Odhlásenie používateľa		Backlog	Backlog	13.10.2011	09.11.2011	
1032	WebTex editor	Feature	Closed	Normal	Zobrazenie zoznamu projektov		Backlog	Backlog	13.10.2011	14.11.2011	
1038	WebTex editor	Feature	Closed	Normal	Možnosť zmazania projektu		Backlog	Backlog	13.10.2011	14.11.2011	
1039	WebTex editor	Feature	New	Normal	Možnosť uploadu projektu		Backlog	Backlog	13.10.2011		
1040	WebTex editor	Feature	New	Normal	Možnosť stiahnuť celý projekt		Backlog	Backlog	13.10.2011		
1042	WebTex editor	Feature	Closed	Normal	Možnosť zmeny mena projektu		Backlog	Backlog	13.10.2011	14.11.2011	
1043	WebTex editor	Feature	Closed	Normal	Nastavenie kolaborantov na projekt a príslušných práv		Backlog	Backlog	13.10.2011	30.11.2011	
1044	WebTex editor	Feature	New	Normal	Možnosť nastavenie hlavného súboru na preklad		Backlog	Backlog	13.10.2011		
1045	WebTex editor	Feature	In progress	Normal	Verziovanie projektu a pridávanie popisov k revíziam		Backlog	Backlog	13.10.2011		
1046	WebTex editor	Feature	Closed	Normal	Zobrazenie jednotlivých revízií súborov v projekte		Backlog	Backlog	13.10.2011	30.11.2011	
1047	WebTex editor	Feature	Closed	Normal	Vytváranie nového prázdneho súboru		Backlog	Backlog	13.10.2011	30.11.2011	
1048	WebTex editor	Feature	New	Normal	Vytváranie nového súboru z templateov		Backlog	Backlog	13.10.2011		
1049	WebTex editor	Feature	New	Normal	Vytváranie nového súboru z templateov		Backlog	Backlog	13.10.2011		
1050	WebTex editor	Feature	New	Normal	Uloženie súboru medzi templaty		Backlog	Backlog	13.10.2011		
1051	WebTex editor	Feature	Closed	Normal	Možnosť zmazania existujúceho súboru		Backlog	Backlog	13.10.2011	30.11.2011	
1052	WebTex editor	Feature	New	Normal	Možnosť uploadu súboru		Backlog	Backlog	13.10.2011		
1054	WebTex editor	Feature	New	Normal	GIT - synchronizácia mimo prostredia webu		Backlog	Backlog	13.10.2011		
1055	WebTex editor	Feature	New	Normal	GIT PUSH		Backlog	Backlog	13.10.2011		
1056	WebTex editor	Feature	New	Normal	GIT PULL		Backlog	Backlog	13.10.2011		
1057	WebTex editor	Feature	New	Normal	GIT autentifikácia menom a heslom alebo pomocou ssh kľúča		Backlog	Backlog	13.10.2011		
1058	WebTex editor	Feature	New	Normal	LaTeX editor		Backlog	Backlog	13.10.2011		
1059	WebTex editor	Feature	In progress	Normal	Syntax highlighting		Backlog	Backlog	13.10.2011		
1060	WebTex editor	Feature	New	Normal	LaTeX autocompletion		Backlog	Backlog	13.10.2011		
1053	WebTex editor	Feature	New	Normal	Možnosť pripájania poznámok k LaTeX kódu pomocou breakpointov		Backlog	Backlog	13.10.2011		
1061	WebTex editor	Feature	New	Normal	Možnosť pdf náhľadu		Backlog	Backlog	13.10.2011		
1062	WebTex editor	Feature	New	Normal	Highlighting aktuálneho riadku		Backlog	Backlog	13.10.2011		
1063	WebTex editor	Feature	New	Normal	Search and replace stringov		Backlog	Backlog	13.10.2011		
1064	WebTex editor	Feature	New	Normal	Full-screen editing		Backlog	Backlog	13.10.2011		
1065	WebTex editor	Feature	New	Normal	Podpora zmeny temy		Backlog	Backlog	13.10.2011		
1066	WebTex editor	Feature	New	Normal	Undo a Redo		Backlog	Backlog	13.10.2011		
1067	WebTex editor	Feature	In progress	Normal	Číslovanie riadkov		Backlog	Backlog	13.10.2011		
1068	WebTex editor	Feature	New	Normal	Informačný status bar		Backlog	Backlog	13.10.2011		
1121	WebTex editor	Feature	New	Normal	Tlacitka na doplnenie kodu		Backlog	Backlog	14.10.2011		

Obrázok 3 – Plnenie Backlogu v čase písania dokumentácie

5.1 Čo umožňuje náš prototyp

Dá sa povedať že tie najdôležitejšie prvky finálnej aplikácie by mali bez väčších problémov fungovať už v tomto prototype. Silné jadro aplikácie je teda pripravené. Ďalšia funkcionalita, ktorá zostala neimplementovaná môže byť vnímaná ako akési vylepšenia aplikácie. Teraz zhrnieme našu funkcionalitu do niekoľkých podkapitol. Každá jedna kapitola predstavuje množstvo práce, ktoré by sa dalo popisovať do väčších detailov. Ale pre prehľadnosť budeme výstižní.

5.1.1 Registrácia a autentifikácia používateľov

Pri práci s našou aplikáciu je zatiaľ potrebná registrácia, neskôr bude určitá funkcionalita sprístupnená aj neregistrovaným používateľom. Každý používateľ sa zaregistruje pod svojím emailom a heslom. Pod týmito atribútmi bude v systéme vystupovať. Máme vytvorenú tabuľku používateľov, kde uchovávame aj iné dôležité atribúty ako čas posledného prihlásenia, IP adresa posledného prístupu, token na reset hesla iné. Tieto atribúty nám umožňujú rozširovať ďalej funkcionalitu. Čiže používateľ sa môže plnohodnotne zaregistrovať a potom sa môže samozrejme aj so svojimi údajmi prihlasovať do aplikácie.

5.1.2 Manažment projektov

Aplikácia umožňuje plnohodnotne vytvárať projekty s ich názvom a opisom. Projekt je vlastne základná jednotka, od ktorej sa všetko dovíja. Pri vytvorení projektu je v súborovom systéme vytvorená zložka s ID používateľa, ktorý projekt vytvoril. V tejto zložke používateľa je podzložka s ID projektu, ktorý vytvoril. Táto zložka projektu je prípravou lokality pre strom súborov, ktoré budú v projekte.
Projektom je možné jednoducho meniť meno, alebo ich opis. Projekty je možné jednoducho zmazať. A kedykoľvek môže používateľ jednoducho vytvoriť novy projekt.

5.1.3 Manažment súborov

Používateľovi sme umožnili vytvárať súbory v nejakom projekte. Tieto súbory môže nielen vytvárať, ale aj meniť ich meno, zmazávať ich, a čo je asi najdôležitejšie, môže editovať ich obsah. Tieto súbory sú potom zapísané v príslušnom projekte na súborovom systéme. Pamätali sme samozrejme na rôzne ošetrenia problémov, ktoré by mohli nastať. Ide napríklad o súbor bez názvu alebo o súbory s rovnakým názvom atď. Aplikácia pri takýchto konfliktoch používateľa upozorní.

5.1.4 Revízie súborov

Vynikajúcim prvkom našej aplikácie je možnosť sledovať zmeny súborov v čase. Na toto jednoduché verziovanie sme využili nástroj Git, ktorý je zakomponovaný v našej aplikácii. Pri každom súbore si teda jednoducho používateľ môže pozrieť jeho predchádzajúce zmeny v čase. Riešenie pomocou verziovacieho nástroja Git je veľmi sofistikované, pretože sú uložené iba zmeny, ktoré nastali, nemusia sa tak ukladať celé súbory v rozličných verziách. Takto sa jednoducho

ukladajú len zmeny a aj tak máme prístup k celej verzii súboru. Táto funkcionalita je tiež nejakou pridanou hodnotou oproti niektorým existujúcim riešeniam.

5.1.5 Kolaborácia používateľov

Veľmi dôležitým prvokom našej aplikácie, ktorým sa asi najviac odlišujeme od podobných riešení, je kolaborácia používateľov na projektoch. Ide o mechanizmus, kedy používatelia môžu spolupracovať. Používateľ môže jednoducho na svojej pracovnej ploche v aplikácii takzvanom Dashboarde vidieť aj projekty, ku ktorým bol pridaný ako spolupracovník. Tieto projekty môže plnohodnotne upravovať a editovať.

Pri kolaborácii je dôležité to, že k svojmu projektu alebo projektu na ktorom spolupracuje môže používateľ jednoducho pridať ďalších spolupracovníkov. Pridávanie spolupracovníkov, prebieha pomocou výpisu emailových adries za sebou oddelených čiarkou. Takýmto jednoduchým spôsobom môžem pridať neobmedzený počet spolupracovníkov iba jednoduchým zoznamom emailových adries. Aplikácia je sofistikovaná a kontroluje, či sú používatelia registrovaný, a teda či nie je zadaná chybná emailová adresa. Aplikácia na rôzne chyby upozorní prehľadnou správou. Pri úspechu je tiež užívateľ upovedomený.

Aplikácia je tak šikovná, že jednotlivým používateľom, ktorí boli pridaní na kolaboráciu pošle informačný email s názvom projektu, v ktorom boli pridaní medzi spolupracovníkov.

Spolupracovníci sú prehľadne zobrazení v tabuľke ku každému projektu. Ktoréhokoľvek spolupracovníka okrem vlastníka projektu možno kedykoľvek jednoducho zmazať.

5.1.6 Autorizácia

Vytvárame našu aplikáciu tak, aby bola použiteľná pre veľké množstvo ľudí. Pri takýchto projektoch je dôležité dbať na bezpečnosť. Musíme preto zabezpečovať prístup k jednotlivým projektom. Prístup k projektu a operácie nad ním by mali byť umožnené iba vlastníkovi projektu alebo spolupracovníkom. A presne takto je to v našej aplikácii zrealizované. Zabezpečujeme preto dôležité informácie pred očami nepovolaných používateľov. Ak by sa iný používateľ pokúšal pristupovať k projektom, na ktoré nemá oprávnenie, napríklad by mohol skúšať zmenu URL adresy, naša aplikácia mu to zamietne a neumožní mu to, presmeruje ho na chybovú stránku.

5.1.7 Editor projektu

Základom dobrej aplikácie kde treba meniť obsah súboru je kvalitný editor. Takýto editor je o to dôležitejší, keď ide o úpravu kódu nejakého druhu počítačového jazyka. Takýmto kódom nepochybne LaTex je. Preto sme chceli aby editor čo najviac uľahčoval prácu používateľovi pri písaní kódu. V tomto prototype sme preto zaviedli do editora číslovanie riadkov a zvýrazňovanie syntaxe, ktoré uľahčujú prácu. Ide o celkom rozsiahlu funkcionalitu editora, ktorá je používateľovi umožnená. Prispôsobujeme si totiž editor otvoreného projektu CODEMIROR. Editor je implementovaný pomocou Javascriptu, je dobre použiteľný, a dá sa rozširovať. Preto by mal byť kvalitným základom pre náš projekt do budúcna, keď budeme zavádzať mnohé vylepšenia.

5.1.8 Stromová štruktúra

Asi netreba príliš vysvetľovať čo to znamená. Súbory by mali byť zobrazené prehľadne, tak aby bolo jasné, ktoré sú nadradené zložky. Práve toto je vhodné realizovať vizualizáciou pomocou nejakého stromu. Túto vizualizáciu rieši náš prototyp. Neskôr by sme chceli implementovať aj úpravu časti stromu.

5.1.9 Preklad LaTex súborov

Tento preklad je asi najväčšou podstatou aplikácie, keďže ide o online editor LaTexu. Náš prototyp aplikácie zatiaľ umožňuje kompiláciu ľubovoľného súboru v projekte do formátu PDF. Po kompilácii sa výsledný súbor uloží a používateľ si ho môže jednoducho stihnúť.

5.1.10 Grafický dizajn a rozhranie

Ide o dôležitý element, ktorý v prvom rade osloví používateľa. Vymysleli sme prehľadný profesionálny a intuitívny dizajn k celej doteraz implementovanej funkcionalite. Dokonca sme pracovali aj na rôznych verziách loga a vytvárali sme ho pomocou vektorovej grafiky, ako pri profesionálnych produktoch vysokého štandardu. Celkové rozhranie aplikácie pôsobí veľmi príjemne, farebné ladenie nie je rušivé a zachováva si veľkú prehľadnosť. Zvolili sme tmavšie farby ktoré sú šetrnejšie k očiam v kombinácii s jemne výraznými farbami, ktoré upozorňujú na dôležité elementy a upozornenia. Dizajn je realizovaný pomocou moderných technológii ako SCSS,

compass, jQury, Javascript. Samotné technológie hovoria za pokrokovosť aplikácie o ktorú usilujeme.

5.2 Zhodnotenie a vízia do budúcna

5.2.1 Zhodnotenie prototypu

Usudzujeme, že prototyp našej aplikácie zvláda naozaj veľa. Priniesli sme mnoho nápadov a nimi môžeme pri kvalitnej implementácii náš produkt postaviť medzi svetové jednotky na trhu. V súčasnom stave už preskakujeme začnú časť konkurencie. Základ našej aplikácie je položený. Naša aplikácia zvláda to čo by používateľ očakával. Podstata toho čo sme chceli zrealizovať je implementovaná. Hrubé múry sú postavené, a mali by byť pevným základom pre našu aplikáciu do budúcna.

5.2.2 Vízia do budúcna

Je dobré, že tá najdôležitejšia kostra je pripravená už v tomto prototype. Iste, vždy je čo vylepšovať, a preto by sme do budúcna chceli vychytať aj najmenšie detaily. Ďalším ťažiskom našej práce by malo byť riešenie problémov ako preklad Latexu v reálnom čase za účelom okamžitého náhľadu. Významnou by mala byť možnosť pridávať poznámky k jednotlivým častiam preloženého dokumentu. Ďalším cieľom bude vylepšiť editor tak, aby poskytoval používateľovi všetko, čo by mohol potrebovať, a čo by uľahčovalo jeho prácu, napríklad automatické dopĺňanie príkazov LaTexu. Ak by sa nám tieto prvky podarilo úspešne zrealizovať, trúfame si povedať, že by naša aplikácia jednoznačne preskočila exitujúcu konkurenciu a stala by sa lídrom na trhu.

6 Úvod do druhého semestra

Doteraz sa vývoj nášho projektu upriamoval na správu používateľov, kolaboráciu medzi nimi, vyriešenie ukladania dokumentov, ich revízii a iných funkcionalít spojených s ponukou služieb zákazníkovi mimo editácie textu. V druhom semestri sme sa zamerali hlavne na vyriešenie zobrazovania zmien dokumentu. Keďže na písaní dokumentu majú pracovať viacerí používatelia a spoločne kolaborovať, bolo nevyhnutné zabezpečiť zvýrazňovanie neakceptovaných zmien dokumentu (neskôr *trackchanges*). Na túto funkcionalitu sme mali v pláne využiť existujúce riešenia, ale nebolo možné ich prepojiť s knižnicami na vývoj editorov ako napríklad *Codemirror* alebo *ACE*. Preto sme boli nútení naprogramovať to sami. Za vzor sme si zobrali existujúce riešenie kolaborácie na dokumente z prostredia *MS Word*, ktoré pracuje presne podľa našich predstáv. V nasledujúcich kapitolách bude podrobne popísaná logika implementácie trackchanges v našom podaní.

Je dôležité ozrejmiť, že správa používateľov programovaná v prvom semestri je jadro projektu a bolo nazvané *WebTex*, do neho je zabudovaný modul (editor) spolu s trackchanges vytváraný v druhom semestri, s názvom *Mirrorlab*. Nasadená aplikácia je integrácia týchto dvoch projektov, ktorá spĺňa požiadavky na editáciu kódu napísaného v *Latexu, trackchanges, revidovanie, správu používateľov a kolaboráciu.*

7 Mirrorlab

Mirrorlab je názov nášho editora určeného na písanie kódu v *Latexu* a *trackchnges*. Práca na ňom pozostávala z dvoch vetiev:

- Zobrazenie zmien v dokumente vykonaných používateľom aktuálne editujúcim dokument
- Zobrazenie zmien od ostatných používateľov

Dokument ako taký je samotne ukladaný pomocou verziovacieho nástroja *Git*, ale pre naše potreby existuje pre každý dokument aj jeho metamodel slúžiaci na uloženie textu spolu s pomocnými značkami označujúcimi zmeny v dokumente.

8 Metamodel

Nasledujúca kapitola popisuje metamodel, jeho využitie, logiku a používanie.

8.1 Popis metamodelu

Metamodel v našom projekte bude dočasná premenná, ktorá je reprezentovaná poľom riadkov. Pri inicializácii editoru je načítaný text rozdelený na základe oddeľovačov riadku ‘\n‘ do poľa riadkov – metamodelu. V metammodeli je udržovaný kompletný text s meta informáciami o type zmien vykonaných nad príslušnými sekvenciami textov. V editore vidíme priamo už len čistý text, bez obaľovacích tagov, ale adekvátne zvýraznený farebne, vzhľadom na meta informácie z metamodelu.

Všetky dôležité operácie, ktoré priamo operujú nad metamodelom sú implementované v súbore `metaModel.js` v triede *`MetaModel`*. Je zrejmé, že aktuálny metamodel, je základným pilierom správneho fungovania aplikácie. Korektný zápis vykonaných zmien do metamodelu je pritom ťažiskom a treba ošetriť množstvo logických problémov.

8.1.1 Štruktúra metamodelu

LaTeX dokument bude vo svojej podstate XML dokument obsahujúci okrem samotného textu aj príznaky, ktoré reprezentujú akciu vykonanú nad určitou časťou textu. Aktuálne rozoznávame dva príznaky – pridaný text (*inserted*) a zmazaný text (*deleted*). Jedná sa teda o tagy,

ktoré obaľujú text, ktorý bol pridaný, resp. zmazaný a navyše obsahujú nasledovné atribúty:
by_user - meno používateľa, ktorý akciu vykonal
date – časová známka akcie
Zvyšný text (text, ktorý nie je obalený tagmi) je text, ktorý bol akceptovaný. Teda ten, ktorý sa bude posielať na transformáciu do PDF súboru. Táto sada tagov sa neskôr bude môcť rozšíriť o komentáre, ktoré bude používateľ môcť pridať k nejakej časti textu.

Príklad dokumentu:

<document>
accepted text
<inserted by_user="user1@gmail.com" date="1.5.2012 8:45">inserted text by user1**</inserted><deleted by_user="user2@gmail.com" date="1.5.2012 9:51"**>deleted text by user2**</deleted>**
<inserted by_user="user2@gmail.com" date="1.5.2012 9:50">inserted by user2**<deleted by_user="kazko@gmail.com" date="1.5.2012 8:45"**>nested deleted tag**</deleted>** continue inserted text **</inserted>**
</document>

8.2 Operácie upravujúce metamodel

Ide najmä o operácie, ktoré priamo upravujú metamodel najčastejšie po volaní z objektu *ModelMerger* v súbore `modelMerger.js`, ktorý pripravuje dáta na zápis, zmazanie, úpravu.
Tu si popíšeme jednotlivé funkcie ktoré priamo upravujú metamodel.
Táto funkcia zabezpečuje vrátenie metamodelu.

```
this.getModel = function(){
  return model;
}
```

Je vhodné využívať volanie:

```
metaModel.getModel()[line]
```

Parameter *line* sa odkazuje na konkrétny riadok, ktorý chceme z metamodelu vrátiť.

Táto funkcia zabezpečuje aktualizáciu konkrétneho riadku v metamodeli, kde je funkcii odovzdávané číslo riadku a upravený text podľa príslušnej operácie.

```
this.updateModel = function (lineNumber, updatedText) {
  model[lineNumber] = updatedText;
}
```

V prípade, že bol v editore stlačený enter, je potrebné vložiť do metamodelu nový riadok na konkrétne miesto, to nám zabezpečí nasledovná funkcia.

```
this.addLineToModel = function (lineNumber) {
  model.splice(lineNumber, 0, '');
}
```

Ak zmažeme nejaký riadok z metamodelu, zabezpečíme to volaním tejto funkcie, ktorej stačí ako parameter iba číslo riadku ktorý treba odstrániť.

```
this.removeLineFromModel = function (lineNumber) {
  model.splice(lineNumber, 1);
}
```

Nasledujúce dve funkcie sú potrebné pre zjednodušenie logiky v triede *ModelMerger*, ktorá pripravuje dáta na zápis. Preto pred úpravou a po úprave sú volané tieto funkcie, ktoré najskôr odstránia tagy *document* a po úprave ich dajú na svoje miesto. Tieto obaľovacie tagy sú nutné pre *parser*.

```
this.rmDocTag = function () {
  model[0] = model[0].substring(10);
  model[model.length - 1] = model[model.length - 1].substring(0,
model[model.length - 1].length - 11);
}

this.addDocTag = function () {
  model[0] = '<document>' + model[0]
  model[model.length - 1] = model[model.length - 1] + '</document>';
}
```

Táto funkcia, je dôležitá na znovu načítanie metamodelu. Načíta sa týmto štruktúra, ktorá eviduje všetky zmeny v dokumente. Je potrebná keď sa pýtame na operáciu, ktorá bola nad znakom vykonaná, alebo autora operácie, či dokonca časovú známku. Preto je dôležité, aby po každej zmene metamodelu bola štruktúra aktualizovaná.

```
this.reloadModel = function (isAcceptRejectOperation) {
  origin = model.join('\n');
  changes = parser.getAllUnacceptedChanges(origin, model);
}
```

Ďalej sa v metamodeli nachádza dôležitá funkcia, ktorej parametrami sú pozícia riadku a pozícia znaku v čistom (neotagovanom) texte.

```
this.authorOfChange = function (pos_line, pos_col) {
}
```

Výstupom funkcie je *HASH*:

```
return {
  'user': user,
  'action': action,
  'positionOnTaggedText': positionOnTaggedText,
  'actualTag': actual_tag
};
```

`user` – vráti autora danej zmeny ak je text akceptovaný je vrátená ‘0‘

`action` – vráti zmenu ‘ins‘, ‘del‘ alebo ak je zmena akceptovaná, tak ‘0‘

`positionOnTaggedText` – vráti pozíciu znaku v otagovanom texte, dôležité pri práci s reťazcami.

`actual_tag` – vráti aktuálny tag prislúchajúci danému znaku, na ktorý sa pýtame.

Táto funkcia má zmyslel najmä pre *modelMerger*, v zvýrazňovaní zmien , keď je potrebné sa pýtať na každý znak, je využitá iná efektívnejšia štruktúra, na ktorú sa môžeme pýtať príslušnou funkciou.

8.3 Príprava zmien na zápis do metamodelu

Po nejakej úprave textu v editore sa postará `GroupOfChangesManager` (kapitola 3) o zgrupnutie súvislých zmien do objektu *change*, ktorý je parametrom pre volanie funkcií v objekte *ModelMerger*, ktorý sa postará o korektnú úpravu a prípravu na zápis do metamodelu.

Aplikačná logika práve vytvorenia korektných dát na zápis do metamodelu je umiestnená v *ModelMergeri*. Je to veľmi podstatná vec, aby dáta v metamodeli boli korektné. Bolo tu množstvo problémov, ktoré bolo potrebné ošetriť. Množstvo situácii, ktoré mohli nastať. Ide o náročné reťazcové operácie, kde treba zvažovať posunutia, prekrývania a mnohé špeciálne situácie.

Popíšeme si funkcie, ktoré pripravujú dáta na zápis do metamodelu:

Dôležitý bude pre nás objekt *change*, ktorý vyzerá nasledovne:

```
var Change = function (){
      this.from = {
            line : 0,
            ch : 0
      };
      this.to = {
            line : 0,
            ch : 0
      };
      this.text;
      this.action;
      this.author;
      this.date;
}
```

from = počiatočná pozícia zmeny zložená z riadku a pozície znaku na danom riadku

to = koncová pozícia zmeny zložená z riadku a pozície znaku na danom riadku

text = text, ktorý bol pridaný/zmazaný

action = akcia, ktorá bola vykonaná: „inserted“ alebo „deleted“

author = meno používateľa, ktorý akciu vykonal

date = časová pečiatka akcie

Do funkcií *ModelMergera* vstupuje štruktúra *change*, ktorá má naplnené *from, to, text, action* atribúty. *ModelMerger* už doplní autora a aktuálny dátum s časom.

Táto funkcia zabezpečuje zápis vloženého textu do metamodelu.

```
This.writeAdditions = function (change) {
}
```

Funkcia je rozsiahla a rieši rôzne problémy, z ktorých spomeniem nasledovné:

- Obaľovanie textu príslušnými tagmi
- Vloženia blokov textu
- Vloženia na koncoch iných zmien
- Vloženia do mojich zmien – stačí iba aktualizácia
- Entery – realokácia modelu

Táto funkcia zabezpečuje zápis zmazaného textu do metamodelu.

```
this.writeRemovals = function (change)
```

Funkcia je rozsiahla a rieši rôzne problémy, z ktorých spomeniem nasledovné:

- Obaľovanie textu príslušnými tagmi (text po zmazaní ostáva v metamodeli, musí sa prečiarknuť)
- Zmazania blokov textu
- Delete na konci riadkov, Backspace na začiatku riadkov
- Odstránenie vnorených zmien v príslušnom delete

Táto funkcia zabezpečuje zápis akceptovaného textu do metamodelu.

```
this.acceptChanges = function (change) {
}
```

Funkcia je rozsiahla a rieši rôzne problémy, z ktorých spomeniem nasledovné:

- Rieši akceptáciu zmien na riadku, podľa akcie vykonáva operácie s textom
- Rieši akceptáciu na viacerých riadkov, kde od *change* potrebuje iba pozíciu *from, to* a *action*
- *Insert* preklápa na akceptovaný neotagovaný text
- *Delete* odstráni kompletne z metamodelu

Táto funkcia zabezpečuje zápis zamietnutého textu do metamodelu.

```
this.rejectChanges = function (change) {
}
```

Funkcia je rozsiahla a rieši rôzne problémy, z ktorých spomeniem nasledovné:

- Rieši zamietanie zmien na riadku, podľa akcie vykonáva operácie s textom
- Rieši zamietanie na viacerých riadkov, kde od change potrebuje iba pozíciu from, to a action
- *Insert* – odstráni zmeny z modelu
- *Delete* – ponechá text a zabezpečí aby text nebol obalený zmazavácimi tagmi

8.4 Testovanie

Nakoľko, udržovať aktuálny metamodel v závislosti od vykonanej operácie a v priamej naviazanosti na text s meta informáciami v metamodeli, ide o náročnú logiku, ktorú treba podložiť testami. Počas implementácie tak náročnej funkcionality sa prichádzalo na množstvo bugov, ktoré

boli opravené a sú pokryté aj príslušnými testami (Obrázok 1 a Obrázok 2), ktoré demonštrujú spôsobilosť aplikácie v tomto štádiu vývoja.

9 Manažment spájania zmien

Nasledujúca kapitola ozrejmuje prácu spojenú s efektívnym predávaním zmien metamodelu.

Jednotlivé zmeny nad dokumentom si uchovávame v objekte metamodel spolu s informáciami, o čase zmeny o autorovi zmeny a type zmeny. Na to, aby sme vedeli narábať s jednotlivými zmenami nad dokumentom, je potrebné vedieť zmenu vytvoriť. Editor, ktorý používame (Codemirror 2.2.1), považuje za jednu zmenu napríklad vloženie jedného znaku. Je samozrejmé, že nemôžeme do nášho metamodelu vkladať informácie o zmene vždy po stlačení hocijakej klávesy. Na určenie, kedy a aká zmena sa má zapísať do nášho metamodelu slúži funkcionalita sústredená v triede *GroupOfChangesManager*.

9.1 Zoskupovanie

Celé zoskupovanie znakov vykonávame nad objektom *GroupOfChangesManager*. Tento objekt má atribút objekt typu *Change*, ktorý reprezentuje aktuálnu zmenu, ktorá sa práve zoskupuje. Pri každej novej zmene sa vyhodnocuje podmienka, či je daná zmena z toho istého sektora ako aj aktuálna zmena. Ak je nová zmena z iného sektora, tak sa celá aktuálna zmena odošle na zápis do metamodelu. Pri zoskupovaní rozlišujeme tieto rôzne prípady:

- jednoduché pridávanie znakov
- vkladanie bloku textu
- jednoduché mazanie znakov, vložené aj v metamodeli
- jednoduché mazanie znakov, ktoré ešte nie sú v metamodeli
- mazanie bloku textu

Na **jednoduché pridávanie znakov** slúži funkcia objektu *GroupOfChangesManager putKeyInBufferForAddition (args, modelMerger).* Táto funkcia má ako argumenty objekt args, ktorý dostáva priamo z editora (vložený text, pozícia od, pozícia do) a objekt typu *ModelMerger*. Táto funkcia zoberie pridaný znak a pridá ho do aktuálnej zmeny, teda do objektu *change* v *GroupOfChangesManager*.

```
GroupOfChangesManager.prototype.putKeyInBufferForAddition = function (args, modelMerger)
{
```

```
    var change = new Change();
    change.from = args.from;
    change.to = args.to;
    change.text = args.text;

    //ked zacnem pisat niekde inde
    if(isInsertionFromDifferentSector(this.change, change) || (this.change != null && this.change.action != INSERT_TAG)){
        processChanges(this, modelMerger);
    }

    if (change.text.length > 1) {
        this.change = change;
        this.change.action = INSERT_TAG;
        processChanges(this, modelMerger);
    } else if (change.text != '') {
        //push one Change
        if (!this.changes.isChangeFromNormalOrder(change)) {
            this.changes = this.changes.incrementPositions(change);
        }
        this.changes.push(change);
        this.changes = this.changes.sortArrayByPosition(change);
        this.change = concatArrayOfChanges(this.changes, CHANGE_TYPE_ADD);
        this.change = findEndOfChange(this.change);
    }

    change = findEndOfChange(change);
    return change
};
```

Ak sa jedná o **vkladanie bloku textu**, tak sa hneď aj celá zmena odošle na zápis do metamodelu. Tu je ale dôležité uviesť, že zmena sa odošle na zápis do metamodelu iba vtedy ak sa jedná o blok textu, ktorý je zložený z viacerých riadkov. Ak sa vloží blok textu, ktorý nepresahuje jeden riadok, tak sa s týmto textom následne pracuje ako keby bol pridaný po jednom znaku. Takéto riešenie bolo nevyhnutné z toho dôvodu, že keď používateľ vkladá znaky po jednom do editora (čiže píše normálnou rýchlosťou), tak niekedy sa stane, že stlačí naraz viacero kláves. Pri stlačení viacerých kláves sa do tejto funkcie dostane z editora vlastne blok textu, čiže by to znamenalo, že by sa aktuálna zmena odosielala do metamodelu vždy viackrát aj pri jednoduchom písaní. Táto funkcia obsahuje viacero pomocných funkcií na prerátavanie pozícií a upravovanie reťazca aktuálnej zmeny.

Pri **jednoduchom mazaní znakov**, ktoré sú vložené aj v metamodeli sa každý znak takisto pridá do aktuálnej zmeny, ale typ operácie má nastavený na mazanie. V tomto prípade je volaná funkcia *putKeyInBufferForRemoval*. **Jednoduché mazanie znakov**, ktoré ešte nie sú v metamodeli je riešené tak, že sa jednoducho odstránia z aktuálnej zmeny. Teda ak niekto zadá text a ten hneď aj zmaže, tak táto zmena sa nezapíše do metamodelu. Pri mazaní bloku textu sa celá táto zmena hneď odošle na zápis do metamodelu.

```
GroupOfChangesManager.prototype.putKeyInBufferForRemoval = function
(editorInst, isDel, modelMerger)
{
    var next = editorInst.getCursor();
    var act = editorInst.getCursor();

    var changeValue = {};
    if (isDel) {
        act.ch = act.ch + 1;
        changeValue = editorInst.getRange(next, act);
    } else {
        act.ch = act.ch - 1;
        changeValue = editorInst.getRange(act, next);
    }

    var change = new Change();
    change.text = new Array(changeValue);
    change.from = act;
    change.to = act;

    if (act.ch == -1 || changeValue == '') {
        change.text = new Array('', '');
    }

    if (isDel) {
        this.changesDelete.push(change);
    } else {
        this.changesDelete.splice(0, 0, change);
    }
    this.change = concatArrayOfChanges(this.changesDelete, CHANGE_TYPE_DELETE);
    this.change = findEndOfChange(this.change);
    return change;
};
```

OnChange, OnKeyEvent

Na to aby sme vedeli narábať s každým jedným vloženým znakom, alebo stlačenou klávesou využívame funkcie editora *onChange* a *onKeyEvent*. Tieto funkcie fungujú nasledovným spôsobom. Bezprostredne po stlačení klávesy sa zavolá funkcia onKeyEvent. Ak po uvoľnení tejto klávesy, nastala nejaká zmena v dokumente, tak sa zavolá funkcia onChange. OnChange má dva argumenty, inštanciu editora a objekt, ktorý predstavuje danú zmena, teda text, ktorý bol pridaný (zmazaný text má túto hodnotu rovnú prázdnemu reťazcu). Dôležité je spomenúť, že po zavolaní funkcie onChange sa ešte raz zavolá funkcia onKeyEvent. Pri zoskupovaní znakov využívame práve tieto tri volania funkcií a to nasledovným spôsobom. Bezprostredne po stlačení klávesy sa zavolá funkcia onKeyEvent, v ktorej sa zavolá naša funkcia na spracovanie stlačenej klávesy *managePressedKey*. Táto funkcia je zameraná na ošetrenie funkcionality po stlačení kláves *Backspace* a *Delete.* Po uvoľnení klávesy sa zavolá funkcia onChange, v ktorej sa zavolá už spomínaná funkcia *putKeyInBufferForAddition*.

Pomocné funkcie

Vo funkcionalite na zoskupovanie zmien sa využíva množstvo pomocných funkcií. Väčšinou sa jedná o funkcie, ktoré vyhodnocujú, či je daná zmena z iného sektora. Napríklad funkcia *isInsertionFromDifferentSector*.

```
function isInsertionFromDifferentSector (bufferedChange, change) {
    var lineOfChange = change.from.line;
    var chOfChange = change.from.ch;

    if (bufferedChange == null) {
        return false;
    }
    //different lines
    if (lineOfChange != bufferedChange.from.line) {
        return true;
    }

    if (chOfChange > bufferedChange.to.ch || chOfChange < bufferedChange.from.ch) {
        return true;
    }
    return false;
}
```

Alebo funkcie, ktoré preusporiadavajú aktuálnu zmenu. Napríklad funkcia, ktorá má za úlohu správne doplniť konečnú pozíciu aktuálnej zmeny *findEndOfChange*.

```
function findEndOfChange (change) {
    var newToCh = change.from.ch + change.text.last().length;
    var newToLine = change.from.line + change.text.length - 1;
    change.to = {line : newToLine, ch : newToCh};

    return change;
}
```

10 Získavanie zmien z metamodelu

Pri každom načítaní nášho editora je potrebné získať z metamodelu všetky zmeny v ňom uložené. Táto funkcionalita je nevyhnutná pre následné zvýraznenie zmien od ostatných používateľov, ktoré predstavujú všetky úpravy textu. Táto kapitola približuje túto funkcionalitu načítania zmien z metamodelu.

10.1 Parser metamodelu

V rámci inicializácie editoru je nutné získať zoznam všetkých ešte neakceptovaných zmien kvôli ich zvýrazneniu. Trieda *MetaModel* obsahuje premennú *changes*, ktorá udržiava tieto informácie vo formáte poľa *Change* objektov. Pre získanie tohto poľa zmien bola implementovaná trieda *Parser*, nachádzajúca sa v súbore *parser.js*. Konkrétne ide o metódu *getAllUnacceptedChanges,* ktorej parametrami sú pôvodný text a *metamodel*.

```
var changes = parser.getAllUnacceptedChanges(origin, model);
```

Kvôli optimalizácii bol formát dokumentu navrhnutý tak, aby sa dal parsovať pomocou DOM funkcií. Preto je najprv nutné pretransformovať originálny text na XML. Na toto nám slúži funkcia *stringToXML*.

```
function stringToXML(labeledString){
        var xmlDoc;
        if (window.DOMParser){
                parser=new DOMParser();
                xmlDoc=parser.parseFromString(labeledString,"text/xml");
        }
        // for IE
```

```
        else{
                xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
                xmlDoc.async=false;
                xmlDoc.loadXML(labeledString);
        }
        return xmlDoc;
}
```

Následne je zavolaná metóda *findChanges*, ktorá rekurzívne prejde celý XML dokument, pričom akceptovaný text si nevšíma.

```
function findChanges(node, model){
        var nodes = node.childNodes;

        for(var i in nodes){
                // Text node element (accepted text)
                var nodeType = nodes[i].nodeType;
                if(nodeType === 3){
                        line += nodes[i].nodeValue.split("\n").length-1;
                        continue;
                }

                // Element node
                if(nodeType === 1){
                        var c =  createObjectOfChange(nodes[i], model);
                        changes.push(c.change);
                        line += c.linesCount;

                        // Element with nested elements
                        if(nodes[i].childNodes.length > 1){
                                findChanges(nodes[i], model);
                        }
                }
        }
}
```

Keď narazí na element *inserted* alebo *deleted*, tak zavolá metódu *createObjectOfChange*, ktorá naplní objekt *Change* dátami a vloží ho do poľa.

```
function createObjectOfChange(node, model){
        var lines = self.removeTags(node).split('\n');
        var linesCount = lines.length - 1;
        var nodeInnerText;

        // node contains children nodes
        if(node.childNodes.length > 1){
                nodeInnerText = self.removeTags(node);
        }
```

```
        else{
                nodeInnerText = self.removeTags(lines[0]);
        }

        var c = new Change();
        c.text = nodeInnerText;
        c.action = node.nodeName;
        c.author = node.getAttribute('by_user');
        c.date = node.getAttribute('date');
        c.from.line = line;
        c.from.ch = self.removeTags(model[line]).indexOf(nodeInnerText);

        // There is more tags in one line or there is only one line of text in tag
        if(linesCount === 0 || linesCount === 1){
                c.to.line = line;
                c.to.ch = c.from.ch + numberOfChars(nodeInnerText)-1;
        }

        return {
                change: c,      // object of change
                linesCount: linesCount          // number of detected lines
        };
}
```

Pri parsovaní, ako i pri generovaní plainTextu, je využívaná metóda *removeTags,* ktorá odstráni XML tagy a vráti samotný text zmeny.

Metóda *numberOfChars* bola implementovaná kvôli pravdepodobnému výskytu znakov '<' a '>' v písanom dokumente. Tieto znaky textarea automaticky transformuje na sekvenciu znakov *<* a *>*. Táto metóda tieto znaky vyhľadá, nahradí ich jedným znakom a potom vráti dĺžku textu. Bez tejto metódy by parser vracal chybné indexy, keby sa tieto špeciálne znaky objavili v texte.

Toto pole *Change* objektov tvorí základnú kostru práce so zmenami. Pre zjednodušenie práce bola implementovaná metóda *get*, pracujúca nad týmto zoznamom zmien.

```
this.get = function(pos, param){
        var results = [];
        var actualObj;
        for(var i = 0; i < changes.length; i++){
                if((changes[i].from.line === pos.line) && (changes[i].from.ch <= pos.ch) &&
(changes[i].to.ch >= pos.ch)){
                        results.push(changes[i]);
                }
        }
        actualObj = results[results.length-1];
```

```
        if(param == undefined)
                return actualObj;
        else
                return actualObj[param];
}
```

Povinným vstupom tejto funkcie je pozícia znaku. Nepovinným vstupom je atribút zmeny, ktorý hľadáme. V prípade, že atribút nie je uvedený, tak metóda vráti celý objekt Change. V opačnom prípade konkrétny atribút, ktorý môže byť napríklad počiatočná alebo koncová pozícia zmeny, text, autor, akcia, alebo dátum.

10.2 Metóda na spájanie zmien pre akceptovanie/zamietnutie zmien

Kvôli efektivite a rýchlemu načítavaniu zmien bolo riešenie navrhnuté tak, že každá zmena obsahuje v jednom riadku aj začínajúci aj ukončujúci tag. Teda v prípade, že používateľ napíše päť riadkov textu, metamodel reálne obsahuje päť tagov *inserted*. Avšak pri akceptácii/zamietnutí tejto zmeny je nutné tieto riadky spojiť dokopy. Pre tento účel bola implementovaná funkcia *prepareAcceptOrReject* objektu *MetaModel* a objekt *Change* reprezentujúci celú zmenu. Akceptovaná zmena následne v editore stratí ofarbenie ak bol označený v metamodeli ako vložený a ak bol označený ako vymazaný tak sa text vymaže.

11 Zvýrazňovanie zmien

Asi najdôležitejšou funkcionalitou z grafického používateľského pohľadu je ofarbovanie textu. Na ofarbenie zmien vytvorených používateľmi okrem práve editujúceho používateľa bolo navrhnutých 20 farieb, kontrastne dobre rozlíšiteľných. Na ofarbenie práve vkladaného textu je použitá príjemná tyrkysová farba. Pri zmazávaní neakceptovaných zmien od iných používateľom je tento text prečiarkovaný.

11.1 Zvýraznenie vytvorených používateľmi okrem práve editujúceho používateľa

Pri inicializácii editoru je potrebné správne ofarbiť zmeny od ostatných používateľov. Preto je táto funkcionalita zavolaná priamo do inicializácie editoru v súbore mirrorlab.js. Celá funcionalita je uložená v súbore *displayChanges.js.* Princíp je nasledovný. Po zavolaní zvýrazňovania zmien ostatných sa do poľa objektov *Changes* načítajú všetky neakceptované zmeny pomocou funkcie nad metamodelom *allUnacceptedChanges().* Následne sa priradia farby všetkým používateľom, aby bola zabezpečená prehľadnosť dokumentu. Farieb je, ako bolo už spomínané, 20 sú priraďované vždy pri načítaní editoru. Z toho vyplýva, že farby užívateľom priraďujeme vždy pri otvorení dokumentu a nie sú natvrdo dané používateľom pri prihlasovaní do aplikácie.

Ak máme priradené farby a sú načítane zmeny, je teraz potrebné ich vyfarbiť aj v editore, na toto je využitá funkcia *Codemirroru markChanges().* Táto funkcia berie ako parametre objekty *from* a *to,* ktoré obsahujú premenné s číslom riadku a pozíciu znaku v riadku. Ako tretí parameter je *CSS* trieda určujúca farbu a pri mazaní aj prečiarknutie. Jej použitie:

```
CM.markText(Changes[i].from, Changes[i].to, color);
```

Táto funkcionalita sa volá nie len pri otvorení dokumentu, ale aj pri akceptovaní/neakceptovaní zmien, aby bolo ofarbenie a text dokumentu aktuálny.

11.2 Zvýrazňovanie zmien editujúceho používateľa

Táto funkcionalita zabezpečuje ofarbovanie zmazaného alebo pridaného textu používateľom, ktorý práve na dokumente pracuje. Pre vložený text je vybraná tyrkysová farba a pre zmazanie neakceptovanej zmeny od iných používateľov bolo zvolené prečiarknutie tohto textu.

11.2.1 Vkladanie textu

Najjednoduchšia časť zvýrazňovania je práve ak používateľ vkladá text. Zavolá sa funkcia *markText()* pri zavolaní funkcie *Codemirroru onChange*. Ak sa jedná o vkladanie textu tejto funkcii sa predajú objekty *from* a *to* a tretí parameter “inserted“, ktorý označuje *CSS* triedu pre vloženie textu.

11.2.2 Mazanie textu

Ak sa pri mazaní textu pohybujeme v rámci našich zmien alebo v akceptovanom texte editor sa správa ako každý iný a zvýrazňovanie logicky nefunguje. To sa zaručí vložením funkcionality zvýrazňovania do nasledovnej podmienky:

```
if (keyEvent.type == "keydown") {
```

kde sa preťaží činnosť tlačidiel delete a backspace, a primárnej funkcionality týchto tlačidiel. Ich primárna funkcionalita je vložená do nasledujúceho bloku kódu:

```
if (keyEvent.type == "keyup") {
       CodeMirror.keyMap.basic.Backspace = "delCharLeft";
       CodeMirror.keyMap.basic.Delete = "delCharRight";
}
```

Pri mazaní textu je dôležité vedieť či mažeme akceptované alebo neakceptované zmeny, pretože ak mažeme vlastnú zmenu je potrebné ju vymazať a ak je zmena iných je potrebné ju prečiarknuť. Na zistenie autora zmeny je využitá funkcia *authorOfChange(riadok, znak)*, ktorá vráti autora zmeny (nie len).

Takisto je nápomocná pri zakázaní mazania textu označeného ako vymazaného iným používateľom a je prečiarknutá. Ak je zmena prečiarknutá pomocou funkcie *Codemirror.stop()* pozastavíme prácu editora a nič sa nevykoná.

V nasledujúcom bloku kódu je činnosť nasledujúca po stlačení klávesy backspace (pre delete je výrazne podobná, z toho dôvodu nie je v dokumentácii uvedená).

```
if (keyEvent.which == 8) {
        if(isBackspaceFromDifferentSector(this.change, change)) {
            processChanges(this, modelMerger);
        }
        if ((this.change == null || this.change.action == DELETE_TAG) &&
(change.from.ch > 0)) {
            authorOfChange = metamodel.authorOfChange(change.from.line + 1,
change.to.ch);
            if (actAuthor !== authorOfChange.user){
                if (authorOfChange.action != "del"){
                    change.from.ch--;
                    editorInst.markText(change.from ,change.to, DELETE_TAG);
                    CodeMirror.keyMap.basic.Backspace = "goCharLeft";
                }
                else if(authorOfChange.action == "del"){
                    CodeMirror.stop();
                    return null;
                }
```

```
            }
        }
```

Nasledujúci kód vykonáva mazanie bloku textu po stlačení backspaceu, logika je rovnaká ako pri mazaní po jednom znaku.

```
        if (this.change == null || this.change.action == DELETE_TAG) {
            if (editorInst.getSelection() != '') {
                var lines =
selectionToMultiLineArray(editorInst.getSelection());
                var groupedChange = new Change();
                groupedChange.from = change.from;
                groupedChange.to = change.to;
                groupedChange.text = lines;
                groupedChange.action = DELETE_TAG;
                modelMerger.writeRemovals(groupedChange);
                if (authorOfAllUnacceptedChanges(metamodel, groupedChange) ===
false){
                    editorInst.markText(groupedChange.from ,groupedChange.to,
DELETE_TAG);
                    CodeMirror.keyMap.basic.Backspace = "goCharLeft";
                    CodeMirror.keyMap.basic.Delete = "goCharRight";
                }
                else{
                    CodeMirror.keyMap.basic.Backspace = "delCharLeft";
                    CodeMirror.keyMap.basic.Delete = "delCharRight";
                }
                return groupedChange;
            } else {
                this.putKeyInBufferForRemoval(editorInst, false, modelMerger);
                return 8;
            }
        } else if (this.change.action != DELETE_TAG) {
            this.removeKeysFromBuffer(change, false);
        }
    }
```

12 Akceptácia a zmietnutie zmien

Operácie akceptovania a zamietnutia nejakej zmeny v texte sa vykonajú po stlačení tlačidla. Pre umiestnenie týchto tlačidiel sme použili knižnicu nad Codemirrorom, ktorá predstavuje

určité používateľské rozhranie. Táto knižnica je sa nazýva Codemirror UI[1]. Jej zdrojové kódy sú voľne dostupné a rieši značné množstvo funkcionality, ktorú by mal mať kvalitný editor. Do tejto knižnice sme teda pridali tlačidla na akceptáciu a zamietnutie jednej alebo aj všetkých zmien.
Ide o vyvolanie jednotlivých metód, ktoré sa vykonajú po stlačení tlačidla:

```
accept_all_changes: function () {
  //alert("Accept all changes");
  this.mirrorLab.modelMerger.acceptAllChanges();
  lockWriting = 1;
  this.mirror.setValue(this.mirrorLab.metamodel.plainText());
  lockWriting = 0;
  new PaintChanges(this.mirrorLab.metamodel, this.mirror);
},
reject_all_change: function () {
  //alert("Reject all changes");
  this.mirrorLab.modelMerger.rejectAllChanges();
  lockWriting = 1;
  this.mirror.setValue(this.mirrorLab.metamodel.plainText());
  lockWriting = 0;
  new PaintChanges(this.mirrorLab.metamodel, this.mirror);
},
accept_change: function () {
  var change = this.mirrorLab.metamodel.prepareAcceptOrReject(this.mirror.getCursor());
  if (change !== undefined) {
    this.mirrorLab.modelMerger.acceptChanges(change);
    lockWriting = 1;
    this.mirror.setValue(this.mirrorLab.metamodel.plainText());
    lockWriting = 0;
    new PaintChanges(this.mirrorLab.metamodel, this.mirror);
  }
},
reject_change: function () {
  var change = this.mirrorLab.metamodel.prepareAcceptOrReject(this.mirror.getCursor());
  if (change !== undefined) {
    this.mirrorLab.modelMerger.rejectChanges(change);
    lockWriting = 1;
    this.mirror.setValue(this.mirrorLab.metamodel.plainText());
    lockWriting = 0;
    new PaintChanges(this.mirrorLab.metamodel, this.mirror);
  }
},
```

[1] https://github.com/jagthedrummer/codemirror-ui https://github.com/jagthedrummer/codemirror-ui

Z jednotlivých metód, ktoré sú vyvolané stlačením tlačidla sa vyvolávajú metódy v modelMergeri, ktorý sa stará o úpravu metamodelu v závislosti od danej operácie. Predstavme si teda funkcie, ktoré majú na starosti akceptovanie a zamietnutie vykonaných zmien.

Táto funkcia zabezpečuje zápis akceptovaného textu do metamodelu.

```
this.acceptChanges = function (change) {
}
```

Funkcia je rozsiahla a rieši rôzne problémy, z ktorých spomeniem nasledovné:

- Rieši akceptáciu zmien na riadku, podľa akcie vykonáva operácie s textom
- Rieši akceptáciu na viacerých riadkov, kde od change potrebuje iba pozíciu from, to a action
- Insert preklápa na akceptovaný neotagovaný text
- Delete odstráni kompletne z metamodelu

Táto funkcia zabezpečuje zápis zamietnutého textu do metamodelu.

```
this.rejectChanges = function (change) {
}
```

Funkcia je rozsiahla a rieši rôzne problémy, z ktorých spomeniem nasledovné:

- Rieši zamietanie zmien na riadku, podľa akcie vykonáva operácie s textom
- Rieši zamietanie na viacerých riadkov, kde od change potrebuje iba pozíciu from, to a action
- Insert – odstráni zmeny z modelu
- Delete – ponechá text a zabezpečí aby text nebol obalený zmazavácimi tagmi

Táto funkcia zabezpečuje akceptáciu všetkých zmien.

```
this.acceptAllChanges = function (change) {
}
```

Funkcia sa stará o akceptáciu zmien v závislosti od toho v akom tagu je text obalený:

- Insert preklápa na akceptovaný neotagovaný text
- Delete odstráni kompletne z metamodelu
- Funkcia prechádza celým textom a vykonáva príslušné úpravy

Táto funkcia zabezpečuje zápis zamietnutého textu do metamodelu.

```
this.rejectAllChanges = function () {
}
```

Funkcia sa stará o zamietnutie zmien v závislosti od toho v akom tagu je text obalený:

- Insert – odstráni zmeny z modelu
- Delete – ponechá text a zabezpečí aby text nebol obalený zmazavácimi tagmi
- Funkcia prechádza celým textom a vykonáva príslušné úpravy

Funkcie k svojej činnosti potrebujú významné súčasti. Jednou z nich je informácia o zmene, ktorá je uskutočnená na danom znaku. Túto zmenu (objekt zmeny) získame volaním:

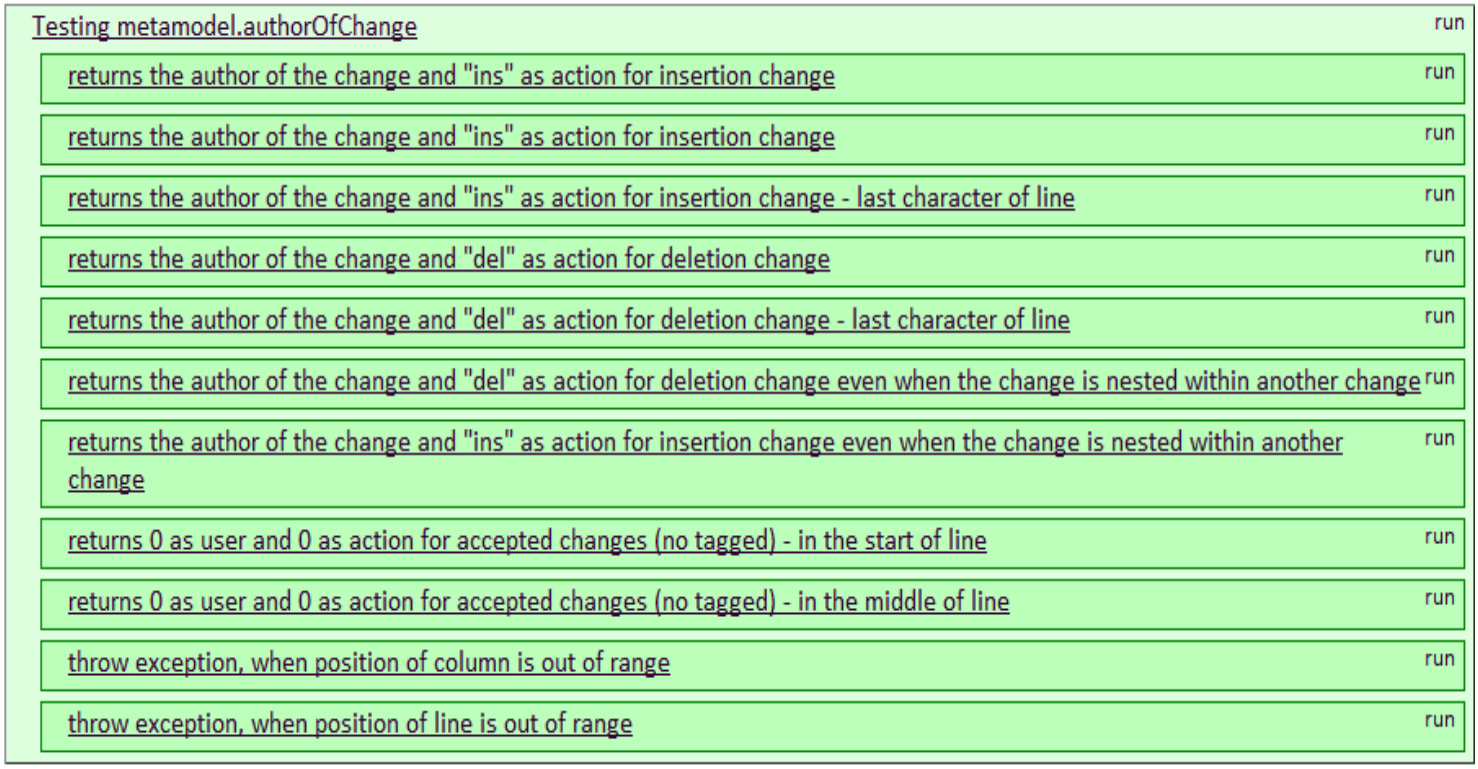
```
var pos = { ch: x, line: y };
var changeObj = metaModel.get(pos);
```

Druhá dôležitá informácia je pozícia znaku v metamodeli, keďže v metamodeli sa nachádzajú aj tagy, ktoré môžu mať rozdielnu dĺžku, a keďže tvorba metamodelu je orientovaná najmä na prácu s textovými reťazcami, je znalosť tejto pozícii nutná.
Príklad získania pozície v otagovanom texte:

```
var fromPositionOnTaggedText = metaModel.authorOfChange(change.from.line + 1,
change.from.ch + 1).positionOnTaggedText - 1;
```

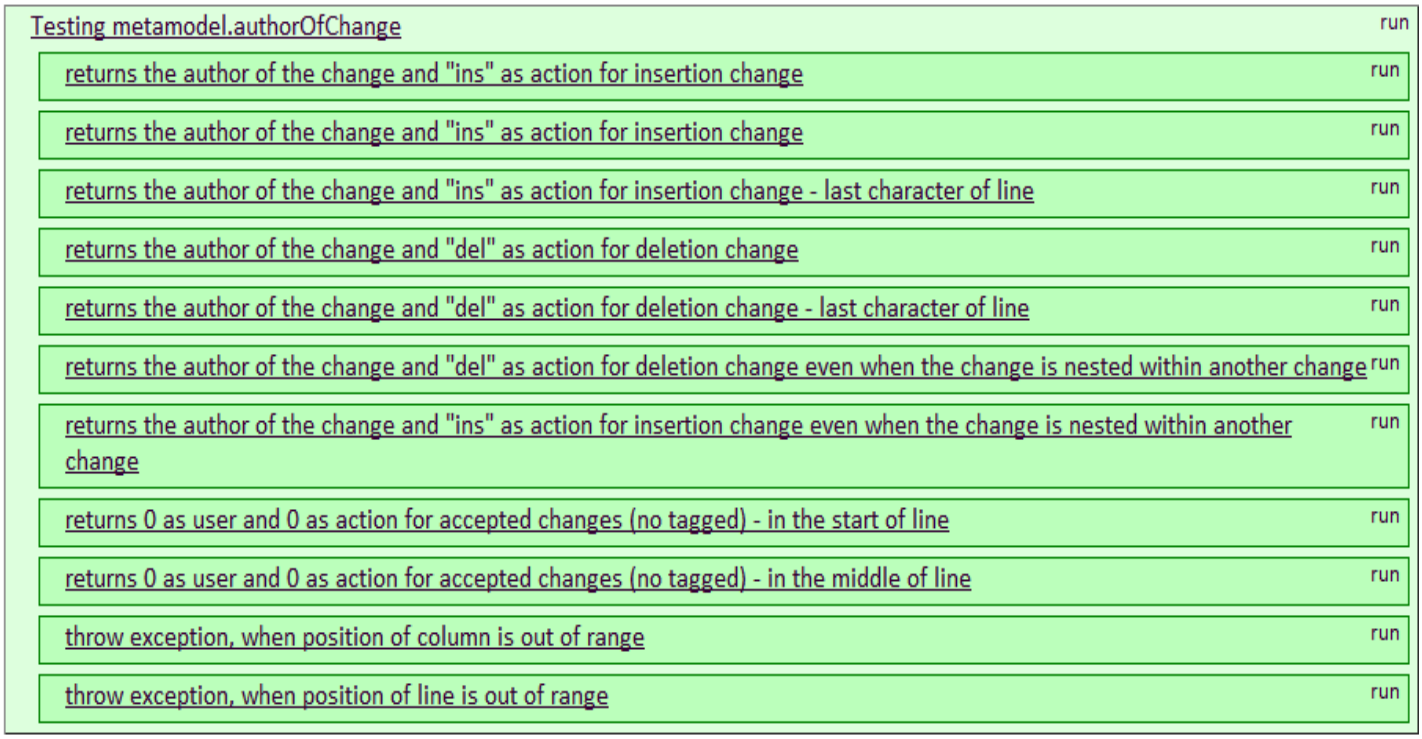
Celkovo práca s metamodelom je veľmi dôležitá vec. Ide o úpravu, vkladanie, mazanie, akceptáciu zamietanie. Je potrebné riešiť množstvo záležitosti ako prekrývanie, vnáranie, zápis do môjho vlastného textu, aktualizácia tagu atd. Aplikačná logika, ktorá súvisí s úpravou metamodelu je preto veľmi náročná a vyžadovala si mnoho uvažovania. Napriek tomu, vždy je priestor robiť to ešte dokonalejšie, nakoľko ide o dosť zložitú problematiku.

13 Prílohy



Obrázok 1 – testovanie funkcie authorOfChange v metaModel.js

Obrázok 2 – testovanie objektu ModelMerger v modelMerger.js



Obrázok č. 3: testovanie funkcionality authorOfChange