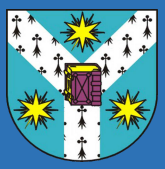

Universitatea Alexandru Ioan Cuza Iași
Facultatea de Informatică
Master Ingineria Sistemelor Software

LUCRARE DE DISERTAȚIE

Multilatex

editor web colaborativ de LaTeX

propusă de

Paul Răzvan **Nechifor**

Sesiunea: iulie, 2014

Coordonator științific
conf. dr. Sabin-Corneliu Buraga

UNIVERSITATEA ALEXANDRU IOAN CUZA IAȘI

FACULTATEA DE INFORMATICĂ

Multilatex: editor web colaborativ de LaTeX

Paul Răzvan Nechifor

Sesiunea: iulie, 2014

Coordonator științific

conf. dr. Sabin-Corneliu Buraga

Declarație privind originalitate și respectarea drepturilor de autor

Prin prezenta declar că lucrarea de licență cu titlul „Multilatex: editor web colaborativ de LaTeX“ este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imagini etc. preluate din proiecte cu sursă publică sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, 24 iunie 2017

Absolvent Paul Răzvan Nechifor

_____

(semnătura în original)

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul „Multilatex: editor web colaborativ de LaTeX“, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea Alexandru Ioan Cuza Iași să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, 24 iunie 2017

Absolvent Paul Răzvan Nechifor

_____

(semnătura în original)

Cuprins

Introducere

Obiectivul acestui proiect este dezvoltarea unui editor/compilator online de documente LaTeX cu suport pentru editarea colaborativă și administrare a istoricului de modificări. Proiectul ar trebui să aibă și alte funcționalități care să ajute editarea colaborativă.

Am ales acest subiect deoarece tipografia și LaTeX se numără printre interesele mele și am ocazia să învăț și să lucrez cu tehnologii pe care nu le-am mai folosit.

Proiectul pe partea de server este scris ca o aplicație web în Node.js folosind JavaScript. Aplicația rulează un serviciu în Upstart care administrează repornirea ei dacă se închide în mod neașteptat precum și alte funcționalități de securitate și logare.

Partea de client folosește multe biblioteci des întâlnite și este scrisă în JavaScript și CoffeeScript (care interoperează bine). Editorul este scris modular (în mai multe fișiere) și se folosește Browserify pentru culegerea dependințelor și unirea într-un singur fișier `.js` care este apoi minificat pentru a-i reduce părțile inutile (comentarii, spații în plus etc).

Toate acțiunile de construire și lansare a aplicației sunt automatizate folosind Gulp pentru a putea integra continuu ușor (de la linia de comandă) modificările făcute proiectului.

Proiectul folosește Vagrant pentru descrierea și provizionarea mașinii virtuale sub care să ruleze serviciul și toate celelalte programe necesare (MongoDB, TeX Live etc). Astfel proiectul poate fi rulat și dezvoltat pe orice sistem de operare și fără a fi necesară instalarea de pachete locale (cu excepția Vagrant și VirtualBox). În plus, în acest fel se prezintă o interfață uniformă fiecărui posibil dezvoltator.

Contribuții

Contribuția mea constă în proiectarea și implementarea întregii aplicații web folosind tehnologii recente care este lansată la `multilatex.com` și al cărei surse este publicată pe GitHub[?].

Funcționalitățile diferite față de alte proiecte similare sunt:

- folosirea unei metode proprii de memorare a istoricului (secțiunea **??**, pagina **??**) folosind un data store propriu (secțiunea **??**, pagina **??**);
- previzualizări dinamice a documentelor prin memorarea tuturor miniaturilor de pagini (secțiunea **??**, pagina **??**);
- notificări pe proiect incluzând comunicare text cu recunoaștere contextuală (secțiunea **??**, pagina **??**);
- și navigarea secțiunilor documentului prin recunoașterea dinamică a acestora (secțiunea **??**, pagina **??**).

Capitolul 1

Sisteme existente

În acest capitol se prezintă funcționarea TEX și LATEX și problemele pe care le au. Apoi se va prezenta cum au rezolvat alții aceste deficiențe folosind tehnologii web.

1.1 TEX

TEX este un program de compoziție tipografică care a fost început în 1977 de către Donald Knuth pentru a permite o compoziție mai bună a articolelor și cărților științifice.

Codul sursă pentru TEX[?] este scris în WEB, un limbaj de programare literară[1] similar cu Pascal. Folosind acest tip de programare codul sursă poate fi compilat într-un fișier PDF[?].

1.2 LATEX

LATEX este un set generalizat de macrouri bazat pe TEX.

Spre diferența de editoarele WYSIWYG, LATEX are avantajul că permite concentrarea pe conținut și nu pe prezentarea lui. În editoare WYSIWYG utilizatorii sunt tentați să folosească spații și linii goale pentru aliniere și page breaking pentru că editează prezentarea în loc de semantică.

Deși era un program bun la conceperea lui pe sistemele limitate de atunci, astăzi LATEX nu exploatează avantajele aduse de sistemele moderne.

[1] Un tip de programare prin care codul sursă este îmbinat cu descrierea lui astfel încât să poată fi citit similar cu o carte.

Unele dintre problemele pe care urmează să le prezint sunt rezolvate de situri precum ShareLatex și WriteLatex. În capitolul următor am să descriu sistemul propriu.

1.2.1 Modul de lucru

LaTeX are un mod de lucru static: editarea fișierelor, compilarea, vizualizarea documentului compilat și repetarea.

LaTeX nu suportă doar recompilarea modificărilor ci necesită recompilarea întregului document ceea ce este încet. Mai rău, sunt necesare mai multe compilări dacă se folosește un cuprins, bibliografie, referințe și altele.

Această problemă nu poate fi rezolvată ușor deoarece ar necesita rescrierea aplicațiilor de bază.

1.2.2 Memorarea stării

LaTeX operează prin folosirea și modificarea unor fișiere în dosarul de lucru. Acest lucru complică orice program care vrea să-l folosească pentru că fișierele generate sunt stocate laolaltă cu cele sursă:

- la curățare (eliminarea fișierelor generate) sunt multe fișiere care trebuiesc șterse;
- după o curățare trebuiesc efectuate un număr nespecificat de compilări până sunt rezolvate toate referințele;
- fișierele sursă nu pot fi prezente într-un data store ci trebuiesc să existe în dosar.

1.2.3 Sistemul de pachete

LaTeX are suport limitat pentru pachete. Nu se specifică versiunea pachetelor când sunt declarate în document, deci compilarea poate duce la rezultate diferite (sau la erori) într-un mod neașteptat. Un model bun în acest caz este Node.js unde există multe modalități de specificare a versiunii preferate și versiuni diferite pot coexista.

În prezent sunt trei distribuții[?] majore de LaTeX cu diferite pachete instalate implicit.

Pentru a avea o compatibilitate cât mai mare, configurarea mașinii virtuale (secțiunea **??**, pagina **??**) depinde de instalarea completă a celei mai noi versiuni de TeX Live (`texlive-full`) de pe Ubuntu.

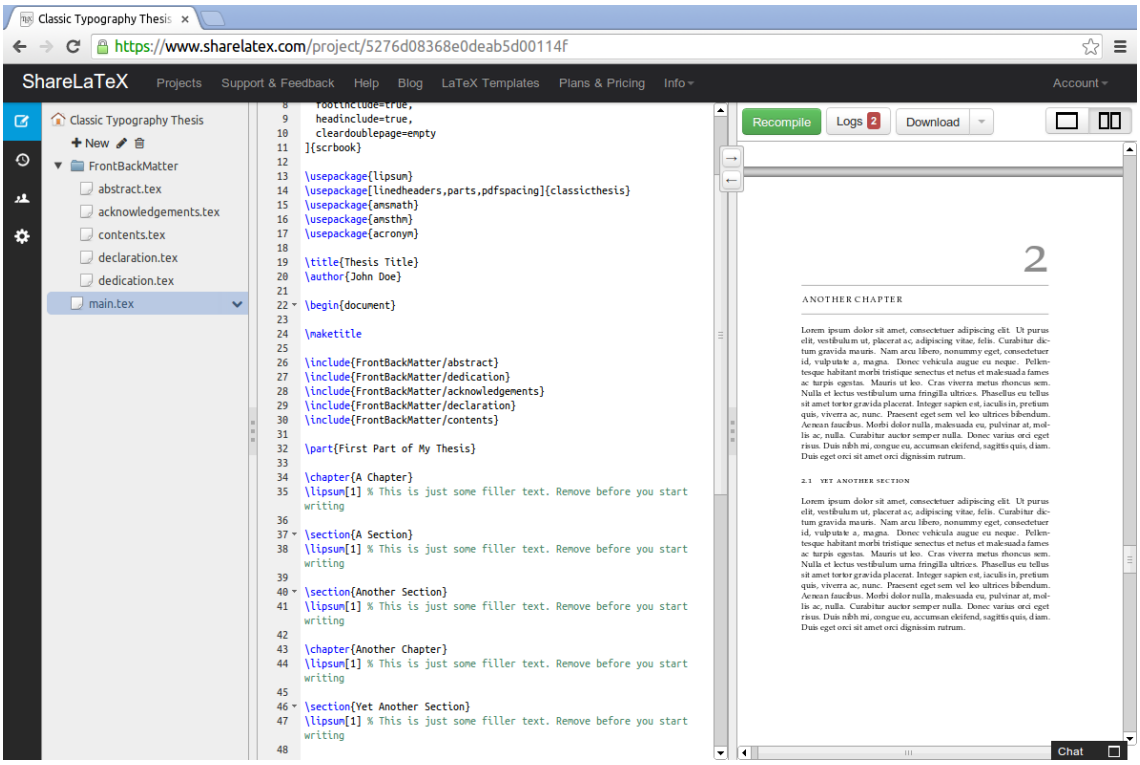

Figura 1.1: Editorul de la ShareLatex.

1.3 ShareLatex

ShareLatex este un o aplicație web de editare a proiectelor LaTeX care a fost lansată în 2013 și a fost bazată pe un proiect experimental din 2011.

De menționat pentru ShareLatex este că:

- este un proiect comercial care oferă o versiune gratis cu mai puține funcționalități;
- codul sursă a proiectului a fost publicat în februarie 2014[?];
- poate face sincronizarea dintre locația de editare (linia din fișier) cu locul unde este afișat rezultatul în PDF;
- folosește PDF.js pentru previzualizarea documentului PDF rezultat;
- și conține un număr foarte mare de șabloane LaTeX.

1.4 WriteLatex

WriteLatex este o aplicație web similară cu ShareLatex cu diferențele principale că:

- poate folosi în loc de editarea textului sursă o variantă numită „text îmbogățit“ care îmbină funcționalități WYSIWYG cu text normal LaTeX;
- folosește randarea paginilor pe server și afișează imagini JPEG pe client (ceea ce reduce din claritatea textului);
- și permite editarea fără înregistrare.

Capitolul 2

Tehnologii folosite

În acest capitol sunt prezentate tehnologiile folosite pentru construirea proiectului și motivele pentru care au fost alese.

2.1 Back-end

Pe partea de server s-au folosit următoarele aplicații/tehnologii:

- **Vagrant**, pentru administrarea mediului de lucru;
- **Upstart**, pentru descrierea serviciului Linux sub care rulează aplicația web;
- **Node.js**, pentru rularea aplicației web;
- **Express.js**, pentru framework-ul web;
- **Jade**, pentru șabloane HTML;
- **Stylus**, pentru generarea CSS;
- și **MongoDB** pentru baza de date.

2.1.1 Vagrant

Dezvoltarea Web, mai mult decât alte tipuri, necesită folosirea a unei multitudini de tehnologii disparate sau strâns legate. Configurarea lor poate fi problematică și a dus la separarea în specializări de dezvoltare și operare IT.

O soluție este să se folosească mașini virtuale cu tot software-ul instalat. Însă acest mod de lucru încetinește pasul de dezvoltare și necesită trafic mult: dacă se schimbă o piesă, durează mult timp până se reconfigurează sistemul virtual și până este transmis la toți dezvoltatorii.

Vagrant[**?**] este un sistem de a lucra cu și de a partaja medii de dezvoltare virtuale care rezolvă această problemă într-un mod diferit. Un sistem virtual este descris într-un singur fișier cu toate proprietățile sale cum ar fi:

- imaginea sistemului de operare de bază (neconfigurat);
- specificarea sistemului virtual (mărime RAM, număr de procesoare etc);
- specificarea rețelei interne (adrese IP folosite, redirectare porturi etc);
- modul de provizionare, fie prin scripturi de sistem sau folosind folosind programe avansate (Chef, Puppet);
- și altele.

Multe imagini de sisteme de operare sunt făcute publice și duplicate local. Prin urmare, transferul unui mediu de lucru nou deseori necesită doar copierea unor fișiere text și reprovizionarea.

În mod normal se instalează sisteme de operare Linux pentru server care rulează fără un afișaj grafic și deci toată comunicarea se realizează prin SSH.

Vagrant folosește implicit VirtualBox pentru virtualizare, dar pot fi folosiți și alți furnizori cum ar fi VMware (pentru care pot fi folosite opțiuni de configurare specializate).

Un alt avantaj este că Vagrant suportă și lansarea pe medii de lucru non-locale cum ar fi Amazon EC2. Acest lucru înseamnă că poate fi folosit pentru dezvoltare, dar și pentru lansare în producție.

Imaginea sistemului de operare pe care o folosesc este Ubuntu 12.04 deoarece era varianta cu suport pe termen lung când am început proiectul.

Pentru provizionare folosesc un script deoarece n-am nevoie de funcționalități mai avansate. În principal provizionarea consistă din:

- actualizarea pachetelor sistemului (pentru securitate);
- instalarea versiunilor cele mai recente de MongoDB și Node de la furnizori (nu din Ubuntu);

- instalarea distribuției LaTeX `texlive-full` care are peste 2 GiB;
- și instalarea altor pachete folosite de aplicație.

2.1.2 Upstart

Programul construit rulează pe backend ca un serviciu în Upstart. Upstart[**?**] este un sistem de administrare a serviciilor pentru Unix bazat pe evenimente. El a fost proiectat ca un înlocuitor pentru init.

Am ales Upstart pentru că este instalat inițial în Ubuntu 12.04. Din următoarele versiuni, însă, Ubuntu și Debian se vor schimba la Systemd care are mai multe funcționalități.

Eu folosesc Upstart prin scriptul care specifică:

- ce utilizator să fie folosit pentru executarea programului (ca să nu fie utilizat `root` pentru siguranță;
- cum să repornească serviciul în caz de erori (se abandonează repornirea în cazul în care eșuează mai multe de 10 ori în 5 secunde);
- spre ce fișier de logare să fie redirectate ieșirile.

2.1.3 Node

Node.js[**?**] este o platformă pentru scrierea de aplicații server care folosește motorul de JavaScript numit V8 folosit și de Google Chrome.

Node a fost proiectat să maximizeze transferul de date prin folosirea de input-output asincron.

V8 poate fi foarte eficient pentru că compilează codul JavaScript în cod nativ (în loc să interpreteze bytecode) și poate să reoptimizeze codul în mod dinamic la rulare în funcție de profilare.[**?**]

Unul dintre principalele avantaje Node este că are un număr de pachete incorporate foarte mic și în schimb se bazează pe managerul de pachete NPM pentru funcționalități non-esențiale.

2.1.4 Express

Express[?] este un framework de aplicații web pentru Node făcut să fie minimal și flexibil. El este modular și nu include funcționalități statice cum ar fi motoare de șablonare unice, dosare statice predefinite etc. O aplicație Express poate chiar fi definită într-un singur fișier.

Deoarece este un nivel foarte mic peste Node, nu reduce din performanța acestuia.

2.1.5 Jade

Jade[?] este un motor de șablonare pentru Node cu sintaxă bazată pe indentare.

Exemplu de cod Jade:

```
mixin authors(list)
  p.authors(class=list.length > 1 ? 'multiple-authors': null)
    each names in list
      span.names= names.join(' ')

#container
  p Primul <em>paragraf</em>.
  .project: +authors([['Paul', 'Nechifor']])
  .project: +authors([['Nechifor'], ['<nume>']])
```

Rezultatul pe care îl produce:

```
<div id="container">
  <p>Primul <em>paragraf</em>.</p>
  <div class="project">
    <p class="authors"><span class="names">Paul Nechifor</span></p>
  </div>
  <div class="project">
    <p class="authors multiple-authors">
      <span class="names">Nechifor</span>
      <span class="names">&lt;nume&gt;</span></p>
  </div>
</div>
```

Funcționalități demonstrate în cod:

- etichetele nespecificate sunt implicit de tip `div`;
- folosirea de cod JavaScript pentru valori;
- se poate folosi și sintaxa HTML standard (`<em>`).

Deoarece Jade are sintaxă cu spații semnificative, ele nu fac parte din codul emis. În rezultatul afișat, spațiile sunt adăugate doar pentru a fi mai ușor de citit, dar în mod normal codul produs nu conține niciun spațiu în plus.

2.1.6 Stylus

Stylus[?] (2010) este un limbaj stylesheet care compilează în CSS. Este asemănător cu Sass (2007) și LESS (2009), dar cu diferența principală că este mult mai succint și este bazat pe indentare. Totuși, Stylus suportă și sintaxa CSS.

Exemplu de cod Stylus:

```
main-color = #f7a
main-size = 18px

a, code
  color main-color
  font-size main-size
  @media (max-width: 479px)
    font-size floor(main-size * 0.8)
  &:hover
    color lighten(main-color, 20%)
```

Rezultatul care va fi produs:

```
a, code {
  color: #f7a;
  font-size: 18px;
}
@media (max-width: 479px) {
  a, code {
```

```
    font-size: 14px;
  }
}
a:hover, code:hover {
  color: #ff92bb;
}
```

În acest exemplu se folosește o variabilă `main-color` pentru culoare care este și manipulată pentru `:hover` (înălbită cu 20 %). Folosind `main-size` mărimea textului pe dispozitive cu ecran mic este micșorată la 80 %. Este demonstrată și imbricarea care evită repetarea: se folosește doar `&:hover` în loc de `a:hover, code:hover`.

2.1.7 MongoDB

MongoDB[?] este o bază de date NoSQL bazată pe documente. Documentele sunt stocate în formatul BSON care este asemănător cu JSON doar că se folosește o stocare binară și are mai multe tipuri de date.

S-a ales această bază de date pentru că avea bibliotecile cele mai mature la începutul proiectării Multilatex.

2.2 Front-end

Pe partea de client am folosit următoarele tehnologii/aplicații:

- **Backbone** pentru paradigma MVC[?] pe client;
- **jQuery** (cu anumite plugin-uri) pentru compatibilitate mai mare pe diferite navigatoare;
- **Ace**, ca editor de cod în pagini web;
- și **PDF.js**, pentru randarea și vizualizarea documentelor PDF.

2.2.1 Backbone.js

Backbone.js[?] este o bibliotecă JavaScript pentru partea de client care furnizează paradigma model–view–controller (MVC).

În principal, Backbone este făcut pentru aplicații cu o singură pagină.

Backbone are o mărime foarte mică. Comprimat, are doar 6,5 KiB. Spre diferență de alte framework-uri, Backbone e făcut să fie foarte simplu și nu este cuplat strâns. Deci poate fi folosit cu multe alte biblioteci fără să necesite controlarea întregului DOM.

2.2.2 jQuery

jQuery este o bibliotecă JavaScript care simplifică programarea pe partea de client și rezolvă parțial lipsa de compatibilitate între diferitele platforme.

2.2.3 Ace

Ace[?] este un editor de text open source făcut de Cloud9 care poate fi incorporat în pagini HTML.

Este făcut specific pentru editare de cod și suport o multitudine de limbaje (printre care și LaTeX).

Editorul este scris în JavaScript și folosește DOM-ul pentru afișare. Ace este succesorul proiectului Mozilla Skywriter care folosea `canvas` pentru afișare.

El este modular și poate fi extins. Datorită acestul lucru am putut să folosesc plugin-ul de la ShareJS pentru editare colaborativă.

2.2.4 PDF.js

PDF.js[?] este o bibliotecă JavaScript care poate să randeze documente PDF pe partea de client fără folosirea de cod nativ. Proiectul este dezvoltat de Mozilla. Pentru randarea paginilor PDF.js folosește eticheta `canvas`.

Printre avantajele lui se numără:

- Este bun pentru securitate pentru că nu se mai execută cod nativ în randarea documentelor PDF. Spre exemplu Adobe Acrobat Reader a avut un număr mare de vulnerabilități.
- Poate fi manipulat din alt cod JavaScript. Un plugin nativ nu poate fi controlat la fel de ușor (spre exemplu folosind eticheta `iframe`) în primul rând datorită faptului că sunt multe platforme și nu toate au un API prin care pot fi controlate.

Un dezvantaj mare este că randarea este prea înceată pentru a nu fi perceptibilă. Acest lucru este agravat la documente foarte mari.

2.3 Comun back-end și front-end

Comun pentru client și server este:

- **Bower** care-i folosit pentru administrarea pachetelor pentru front-end;
- **WebSocket** prin care se face comunicarea dintre editor și server;
- și **ShareJS** care-i folosit pentru editarea colaborativă a fișierelor.

2.3.1 Bower

În variantele precedente am inclus codul pentru modulele necesare (Ace, PDF.js etc) direct în codul sursă a proiectului. Acest lucru duce la supradimensionarea proiectului cu cod care nu-mi aparține. Pentru foarte mult timp acest mod de lucru era normal pentru că nu exista un mod de a administra pachetele front-end.

Bower[?] este un astfel de manager de pachete pentru web. Spre diferență de NPM și altele Bower nu este un manager de pachete pentru un limbaj specific. El funcționează pentru orice tip de fișiere: JavaScript, HTML, CSS și altele. În acest caz Bower definește un pachet ca orice colecție de fișiere și lasă modul de folosirea a codului pentru utilizatori. Bower se folosește de repozitorii de tip Git pentru răspândirea fișierelor.

În acest fel am putut să mut codul pentru dependințele de front-end în afara proiectului. Un avantaj aici este că pot să am un istoric separat. Dacă trebuie să actualizez la un pachet mai nou nu trebuie să includ tot codul modificat ci doar trebuie să specific versiunea nouă în fișierul de configurare Bower și să fac actualizarea locală.

2.3.2 WebSocket

WebSocket[?] este un protocol de comunicare full-duplex în principal implementat pentru comunicarea cu navigatoare web. Specificația WebSocket a fost standardizată în 2011 și face parte din grupul de funcționalități HTML5.

WebSocket are mai multe avantaje peste versiunea de comunicarea Comet sau Ajax:

- se reduce traficul și latența deoarece nu trebuie retrimis tot antetul HTTP de mai multe ori;
- se poate face comunicare bidirecțională printr-o singură conexiune;
- permite trimiterea de mesaje text sau binare, nu doar de flux de caractere ca la HTTP.

În plus, datorita faptului că se folosește tot portul 80 se păstrează caracteristicile de securitate (el fiind rar blocat).

În prezent, toate navigatoarele moderne suportă protocolul WebSocket.

2.3.3 ShareJS

ShareJS[?] este framework pentru editarea colaborativă a fișierelor text. Este compus dintr-o bibliotecă pentru partea de client și un pachet pentru Node.

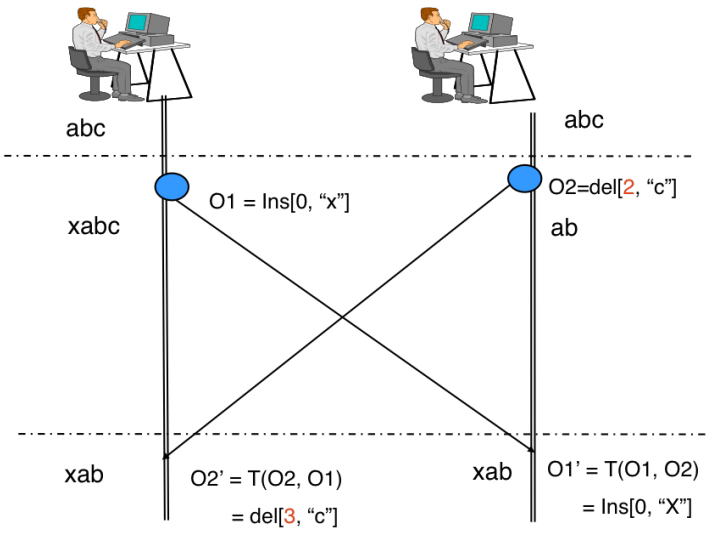

Figura 2.1: Exemplu de comunicare OT.[?]

Pentru permanența fișierelor, ShareJS folosește mai multe tipuri de baze de date.

În mod normal comunicarea dintre client și server se poate face prin WebSocket, dar dacă nu este disponibilă se poate face și folosind comunicarea Comet.

Pentru menținerea modelului de consistență este folosită tehnologia operational transformation (OT).

OT a fost inventat în 1989 de către C. Ellis și S. Gibbs și a fost folosit pentru prima dată în sistemul GROVE[**?**].

OT este o tehnologie de menținere a consistenței în sistemele colaborative și inițial a fost folosit doar pentru fișierele text după care a fost extinsă pentru funcționalități mai avansate (undo, blocare) și documente structurate (non-text).

În figura **??** este demonstrat un mic exemplu de comunicare. Se presupune un document cu textul „abc“. Un utilizator vrea să adauge „x“ la început și altul să șteargă „c“. Documentele lor diferă până când își comunică modificările. Deoarece fiecare știe starea documentului când s-a făcut modificarea celuilalt aplicând transformările se ajunge la același document.

Capitolul 3

Sistemul propus

În acest capitol se prezintă proiectarea Multilatex și se explică în detaliu fiecare componentă.

3.1 Proiectare

În scrierea sistemului propriu am început întâi cu proiectarea pe care am descris-o pe situl proiectului la `multilatex.com/blog` . Urmează principalele probleme asupra cărora a trebuit să mă decid.

3.1.1 Documente mari

LaTeX suportă includerea fișierelor `.tex` în alte fișiere `.tex`. Prin urmare, un mod de lucru cu documente mare este să fie separate în fișiere pe bază de capitole sau secțiuni.

Deși această variantă este suportată în Multilatex, poate fi problematică din unele puncte de vedere: când un document este la început structura lui nu este foarte clară și se recurge la mutarea de capitole, paragrafe etc. între diferitele fișiere.

Principala problemă la a folosi un fișier monolitic este greutatea de navigare, deci am ales să rezolv problema asta prin a implementa în panoul arborelui de fișiere un nivel suplimentar prin care fișierele `.tex` listează toate capitolele, secțiunile și subsecțiunile. Astfel se poate sări ușor printre părțile documentului.

3.1.2 Editarea colaborativă

Există două modalități de editare colaborativă: în timp real (precum Google Docs) sau bazată pe commit-uri (precum Git sau alte sisteme de control a versiunilor).

Deoarece LaTeX nu funcționează precum un editor WYSIWYG, a doua variantă pare mai bună, dar operațiile de integrare ar complica mult sistemul.

Sunt probleme și la colaborarea în timp real. Aici se complică operațiile de anulare (undo) deoarece nu există un istoric bine definit. În plus, nu se poate decide ușor care sunt versiunile intermediare.

Am ales până la urmă să le implementez pe ambele variante. Prima folosind ShareJS și a doua folosind MongoDB și un file store propriu. În retrospectivă, acest lucru a complicat mult sistemul.

3.1.3 Compararea modificărilor

O altă problemă pe care mi-am pus-o la proiectarea a fost compararea versiunilor intermediare și contribuțiilor de la fiecare persoană. Folosind doar commit-uri acest lucru este ușor, dar cu versiune în timp real este greu să fie revizuite contribuțiile fiecărei persoane deoarece ce se pot suprapune foarte mult.

Ca soluție am decis ca să permit commit-uri cu mai mulți autori. Astfel când se face un commit, fiecare persoană care a făcut o modificare între timp va apărea ca un autor. Dezavantajul este că nu se poate stabili pentru o versiune ce modificare a făcut fiecare dintre autori, deci trebuiesc făcute commit-uri dese.

3.1.4 Compilatorul de LaTeX

Am considerat și folosirea unui compilator de LaTeX care să genereze PDF-urile pe partea de client. Spre exemplu `texlive.js`[?] poate compila proiecte direct din navigator. Acest lucru este posibil pentru că TeX Live a fost compilat în JavaScript folosind Emscripten.

Problemele majore sunt că este prea încet și nu suportă toate pachetele TeX Live. Acest lucru chiar ar fi imposibil deoarece o instalare completă a `texlive-full` are mai mult de 2 GiB.

3.1.5 Importare și exportare

Un factor important pentru succesul unui astfel de sit este compatibilitatea și încrederea în permanența conținutului, deci trebuie să existe un mecanism ușor de a importa și de a exporta proiectele. Acest lucru este parțial complicat de faptul că proiectele LaTeX pot avea o multitudine de fișiere. Pentru a rezolva această problemă am ales să consider că fișierul denumit `main.txt` sau singurul fișier `.tex` este cel principal.

3.1.6 Altele

Alte probleme au fost legate de tehnologiile pe care să le folosesc pentru construirea proiectului. Aceste programe/biblioteci vor fi prezentate mai departe în acest capitol.

3.2 Structura aplicației web

În ceea ce urmează am să descriu cum am structurat paginile, conținutul și organizarea sitului.

Consider că audiența principală a sitului va fi construită din persoane care au mai folosit LaTeX în trecut și doresc să se mute în cloud și persoane care vor să colaboreze mai bine în editarea de proiecte LaTeX.

3.2.1 Prima pagină

Rolul primei pagini este de a explica într-un mod foarte rapid ce se poate face cu acest sit și să-i convingă pe potențialii utilizatori să se înscrie. Pentru asta am făcut posibil o înscriere cât mai rapidă (vezi secțiunea **??** la pagina **??**).

Principala funcționalitate este un jumbotron cu o demonstrație și formularul de înregistrare rapidă.

Alte lucruri de luat în considerare ar fi recomandări de la alți utilizatori.

3.2.2 Galerie

Galeria prezintă cele mai populare proiecte. Într-un fel, rolul ei este de a lista cele mai bune creații și de a demonstra ceea ce este posibil cu acest proiect. Pagina este structurată ca o grilă de miniaturi.

Luând inspirație de la Speaker Deck, am decis să folosesc și eu o previzualizare dinamică a proiectelor. În loc să folosesc o pagină predefinită ca miniatură pentru un document, la compilarea unui commit fac miniaturi la toate paginile și mișcarea pe orizontală asupra miniaturii decide care pagină să fie afișată. Astfel se poate previzualiza ușor un document fără să fie deschis.

Tot aici sunt listate și șabloanele precompilate (secțiunea **??**, pagina **??**) de unde pot fi și duplicate ușor.

3.2.3 Utilizator

Nu am intenționat să fac un sit social, deci pagina unui utilizator nu conține decât metadate publice și listări a proiectelor și activității lui.

3.2.4 Proiect

Pagina proiectului conține o prezentare generală a proiectului cu toate modurile principale de controlare a lui.

În mod proeminent este afișată lista de butoane simple pentru:

- accesarea editorului de proiect;
- vizualizarea documentului PDF rezultat
- duplicarea proiectului;
- și descărcarea proiectului în format ZIP.

Duplicarea

Duplicarea proiectului nu este costisitoare pentru server (explicat la secțiunea **??**, pagina **??**) și din acest motiv ea este încurajată.

Dacă proiectul a fost duplicat din altul, este afișată legătura către proiectul părinte.

La secțiunea **??** de la pagina **??** se discută cum poate fi folosită duplicarea pentru editare colaborativă.

Descărcare

Cu butonul de descărcare se poate lua doar versiunea curentă a proiectului. Descărcarea versiunilor vechi poate fi făcută din istoric. La fel și duplicarea sau vizualizarea versiunilor vechi.

Colaboratori

Din pagina proiectului se poate face și administrarea setului de colaboratori (cei care pot edita proiectul curent în totalitate). Implicit, doar creatorul este listat și doar el poate să adauge și să șteargă alți utilizatori.

Intuitiv nu are sens să fie afișat și autorul deoarece nu poate fi șters, dar așa se poate afla instant că acolo este lista de colaboratori și de acolo pot fi adăugați.

3.2.5 Istoric

Pagina de istoric a unui proiecte listează toate versiunile sale. O versiune afișează:

- numele versiunii și legături către pagina versiunii și opțiuni de descărcare;
- lista de autori a versiunii (cei care au făcut o modificare oricât de mică de la ultima versiune);
- un rezumat al modificărilor făcute asupra proiectului (ce fișiere au fost modificate, șterse etc)
- și alte metadate scurte.

Pagina unei versiuni vechi este aproape identică cu pagina proiectului la starea la care s-a făcut commit-ul.

3.2.6 Fișiere

Listarea fișierelor este prezentă pagina proiectului, pagina de versiune și în listarea conținutului fișierului însăși.

Pentru listarea fișierelor este folosită colorarea sintactică pe parte de client folosind o bibliotecă JavaScript. Pe lângă lângă recunoașterea fișierelor text mai sunt afișate și un număr de imagini suportate de LaTeX.

3.2.7 URL-uri

Cred că proiectarea URL-urilor este importantă pentru că ajută la folosirea mai ușoară a sitului. Un exemplu de situri cu URL-uri bune este GitHub unde dacă numele meu de utilizator este `paul-nechifor` și numele la un proiect este `multilatex` URL-ul pentru el este `github.com/paul-nechifor/multilatex`. Ceea ce este ușor de memorat și ușor de transmis în medii non-electronice. A se compara cu alte variante des întâlnite precum un ipotetic `github.com/projects.php?user=51234&project=654742`.

Urmează exemple a URL-urilor folosite unde `:nume` reprezintă o variabilă.

`/` Pagina principală.

`/api/...` Toate paginile de folosite de API.

`/blog` Cele mai recente articole din blog.

`/blog/:post` Un articol specific.

`/explore` Pagina de explorare a galeriei de proiecte.

`/:username` Pagina personală a unui utilizator.

`/:username/:location` Pagina unui proiect.

`/:username/:location/commit/:n` Pagina unui proiect la o anumită versiune (commit).

`/:username/:location/commit/:n/fork` URL-ul de duplicare a unui proiect la o anumită versiune.

`/:username/:location/head/pdf` Vizualizarea documentului compilat (varianta curentă din editor).

`/:username/:location/head/log` Vizualizarea logului de compilare.

`/:username/:location/zip` Descărcarea proiectului în format ZIP.

3.2.8 Dispozitive mobile

Datorită creșterii utilizării dispozitivelor mobile pe web (figura **??**) este important să fie luate în considerare.

Pentru realizarea sitului am utilizat o proiectare web adaptabilă (responsive) folosind interogări `@media`. Singura excepție este pagina editorului deoarece necesită un ecran mare. Pot fi observate diferențele în figura **??** (desktop) și figura **??** (mobil).

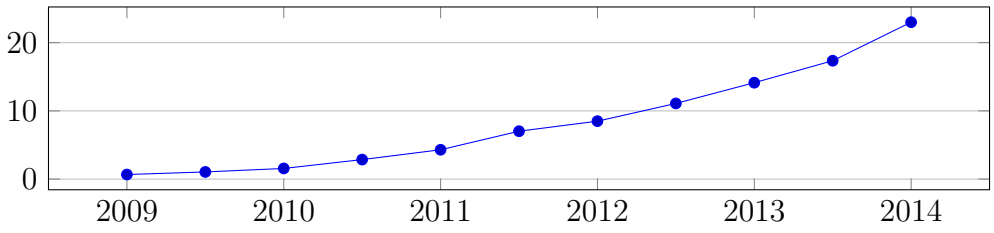

Figura 3.1: Utilizarea dispozitivelor mobile pe Web (ca procent)[**?**].

3.2.9 Stil

Stilul sitului nu se numără printre cele mai importante aspecte și am ales să folosesc Bootstrap 3 cu puține modificări pentru o estetică plată și simplă.

3.3 Integrare continuă

Această practică se referă la integrarea de mai multe ori pe zi a modificărilor din ramura principală.

Kent Beck și Ron Jeffries au inventat integrarea continuă odată cu Extreme Programming în 1997. Kent Beck a publicat o carte în care detaliază integrarea continuă în 1998[**?**].

Printre principiile acestei metodologii se numără:

- folosirea unui sistem de versionare (pentru sistemul propus s-a folosit Git).
- automatizarea construirii programului (aici s-a trecut prin trei versiuni și s-a rămas la folosirea Gulp care este discutat la secțiunea **??** de la pagina **??**);
- accesul tuturor dezvoltatorilor la ramura principală;

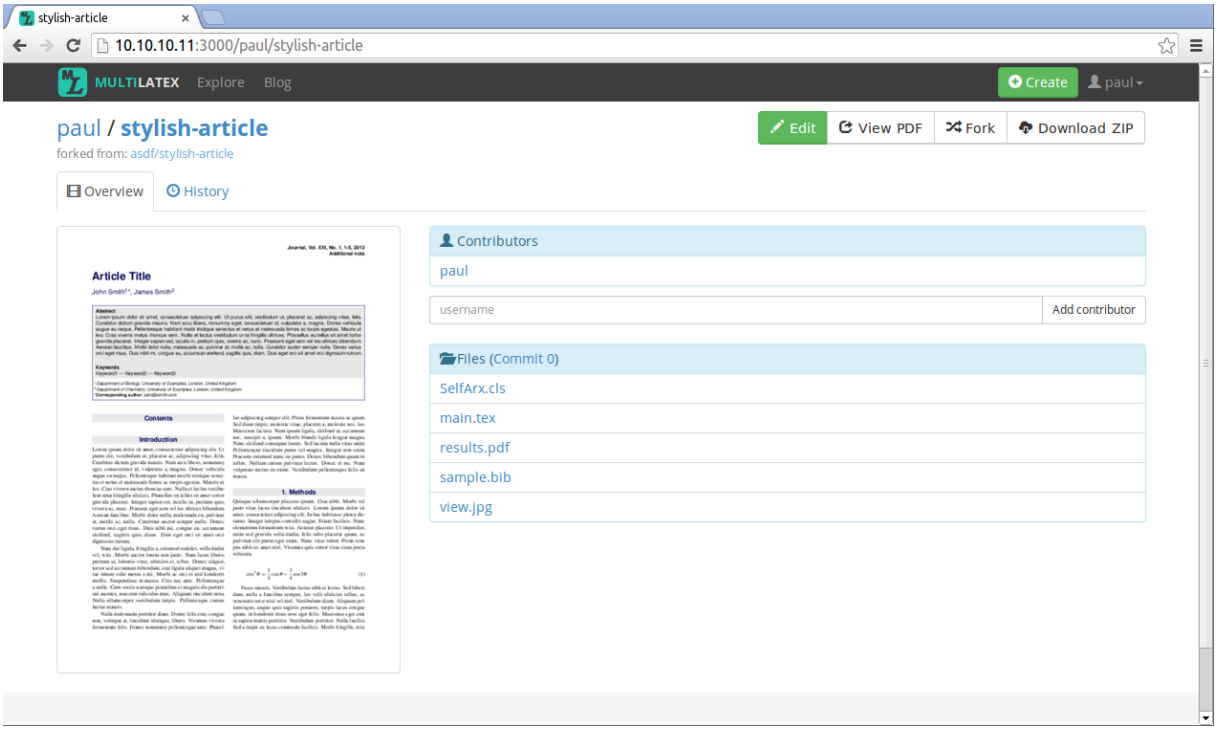

Figura 3.2: Situl vizualizat în format pentru desktop.

- automatizarea lansării (acest lucru este descris tot în Gulp)
- și altele.

3.4 Construire

Construirea proiectului a susținut cele mai mari modificări și a trecut prin trei variante. Prin construire mă refer la compilarea, configurarea și instalarea părților necesare.

3.4.1 Programul propriu

Inițial am început prin a scrie propriul program pentru a face construirea. Programul folosea pachetul `commander` pentru a procesa argumentele (spre exemplu pentru a suprascrie opțiuni din fișierul de configurație) și avea opțiunile de a instala și de a face lansarea.

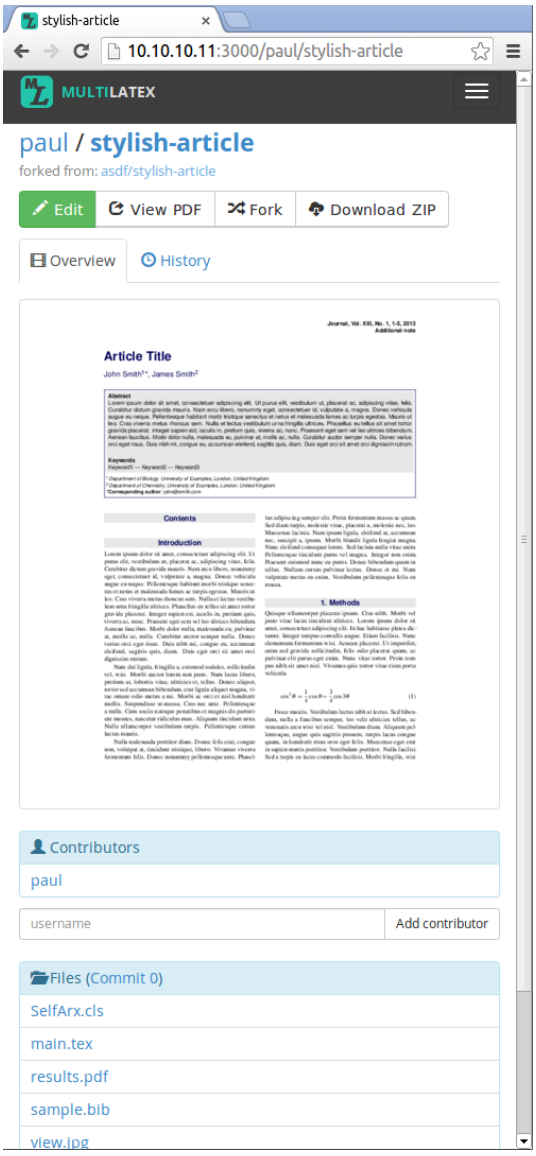

Figura 3.3: Situl vizualizat în format pentru dispozitive mobile.

Instalarea implica copierea fișierelor prin `rsync`, crearea dosarelor speciale și repornirea serviciului Upstart. Atunci nu făceam compilare de niciun fel.

Lansarea implica copierea dosarului de dezvoltare pe un sistem specificat și rularea scriptului de instalare.

După asta am descoperit Grunt care rezolvă această problemă într-un mod general.

3.4.2 Grunt

Grunt[?] este un sistem de automatizare a proceselor de lucru cu foarte multe pluginuri care funcționează prin descrierea de sarcini de lucru.

Un avantaj peste folosirea de fișiere `Makefile` este că Grunt este scris în JavaScript și modulele sunt făcute să fie independente de platformă.

Compilare Stylus

Fișierele Stylus pot fi compilate la runtime de Express, dar este mai eficient să fie compilate dinainte în CSS.

Unificare JavaScript

În mod normal fișierele JavaScript sunt incluse individual in pagini prin etichete `script`. Acest lucru obligă trimiterea unei cereri pentru fiecare fișier. În HTTP 1.1, nu mai este necesară crearea unei conexiuni noi per cerere, dar tot există overhead pentru cereri.

O altă problemă este lipsa de modularizare. Un modul logic compus din mai multe fișiere JavaScript trebuie inclus în fiecare pagină. Astfel pot exista coliziuni de variabile și includerea etichetelor `script` în ordine greșită poate duce la erori.

Pachetul Browserify[?] permite compunerea fișierelor JavaScript într-unul singur folosind specificația CommonJS de descriere a modulelor (cea folosită și de Node). Astfel, în loc să fie descris un pachet în mod liniar prin etichete `script` în fiecare pagină în care este inclus, fiecare fișier descrie ce dependințe are și Browserify compune tot codul necesar în funcție de fișierul de intrare și dependințele acestuia în mod recursiv.

Un dezavantaj la acest mod de lucru este că se complică procesul de depanare deoarece nu mai corespund liniile la runtime cu liniile din fișierele proiectului. Din fericire acest lucru este rezolvat prin folosirea de mapări la sursă (source maps) unde

pe lângă codul modulului unificat este inclusă în comentarii și maparea la codul inițial. Însă recunoașterea mapărilor este prezentă doar în navigatoarele recente.

Minificare

Pentru fișierele JavaScript și CSS se poate realiza o optimizare prin care se scot toate spațiile și comentariile inutile. Operații mai avansate de optimizare reprezintă redenumirea variabilelor locale la JavaScript sau eliminarea de cod nefolosit (cod după `return` în JavaScript sau opțiuni suprascrise de mai multe ori la CSS).

Instalarea și lansarea

De data aceasta pașii pentru instalare sunt descriși în JavaScript. După compilarea fișierelor, ele sunt copiate în dosarul necesar (folosind `rsync` pentru evitarea copierii fișierelor neschimbate) și se execută restul pașilor (crearea dosarelor și utilizatorilor dacă e nevoie, repornirea serviciului). Lansarea este similară variantei precedente.

Altele

Alte sarcini descrise sunt de compilare a șabloanelor de proiecte LaTeX și listarea fișierelor de logare în timp real de pe mașina țintă.

Problema

Problema la Grunt este că preferă configurația înainte de cod. Acest lucru duce la configurații mari și organizate pe operații în loc de ordinea logică a construirii.

3.4.3 Gulp

Gulp[**?**] este un sistem de automatizare similar, dar cu diferența principală că sarcinile sunt descrise prin fluxuri de transformare și se preferă cod înainte de configurații stufoase.

Astfel construirea proiectului poate fi descrisă printr-un graf de dependințe și transformări din care se execută doar cele necesare.

CoffeeScript

A treia iterație a fost și momentul în care am început să folosesc CoffeeScript. Însăși fișierul de configurare Gulp este scris în acest limbaj. Browserify poate fi făcut să recunoască fișiere CoffeeScript și să le compileze înainte de unificare.

Bower

Începând cu mutarea la Gulp am început să folosesc și Bower (discutat la secțiunea **??**, pagina **??**) pentru administrarea pachetelor pentru front-end.

3.5 Autentificare

Stocarea incorectă a parolelor încă este o problemă des întâlnită astăzi și a dus la compromiterea a milioane de conturi pentru multe companii (vezi tabelul **??**).

Eu am folosit hashing a parolei cu salt (data înregistrării). O altă variantă ar fi fost bcrypt, dar verificarea este mai computațional intensivă.

Data	Compania	Parole unice	Parole totale
2012-07-11	Yahoo	342.508	442.832
2012-02-22	YP	833.994	1.566.156
2009-12-04	RockYou	14.344.173	32.603.145
2011-12	CSDN	4.037.902	6.428.632

Tabela 3.1: Compromiterea parolelor stocate în clar[**?**].

3.5.1 Sugestii

Unele situri folosesc sau chiar obligă folosirea de sugestii pentru amintirea parolelor. Eu nu am făcut asta deoarece în cazul compromiterii bazei de date acest lucru ajută la găsirea parolelor. Acest lucru s-a întâmplat în 2013 când hash-urile și sugestiile pentru 150 de milioane de conturi Adobe au fost publicate. Deoarece nu se folosea salt, dacă mai mulți utilizatori aveau aceeași parolă, sugestiile ajută la ghicirea parolei.

3.5.2 Părți terțe

O variantă de autentificare este folosirea unor părți terțe. Eu nu am făcut asta pentru că acest lucru implică dependența de alte sisteme care pot încetini dezvoltarea rapidă.

3.5.3 Înregistrare rapidă

În loc să permit autentificarea cu conturi de pe alte situri, am ales să permit înregistrarea foarte rapidă. De pe prima pagina se poate realiza înregistrarea scriind doar numele dorit și parola (și confirmarea ei).

3.5.4 Probleme

Deoarece nu am un certificat, încă nu folosesc HTTPS pe `multilatex.com`, deci autentificarea nu este foarte sigură.

3.6 Izolare LaTeX

TeX este un limbaj puternic și deci posibil periculos. Trebuie avut grijă când se execută cod arbitrar de la surse necunoscute.

Am scris un modul care să izoleze (sandboxing) pe cât posibil compilarea proiectelor LaTeX.

3.6.1 Execuția comenzilor de sistem

Una dintre primele probleme este că LaTeX poate să execute comenzi de sistem arbitrare folosind construcția `\write18{comandă}`.

În mod normal acestă construcție nu este disponibilă, dar pentru mai mare siguranță poate fi oprită direct din execuția `pdflatex` cu argumentul `-no-shell-escape`. Se mai folosește în plus argumentul `-halt-on-error` pentru a se opri execuția programului în loc să aștepte corectarea interactivă a erorilor.

3.6.2 Cod rău-intenționat

TeX este un limbaj Turing-complet[?] deci nu se poate determina fără a fi fie executat codul dacă compilarea documentului se va termina.

LaTeX, cu toate pachetele sale, este uriaș, deci foarte probabil există probleme nedescoperite care pot duce la umplerea memoriei sau intrarea în bucle infinite.

Problema terminării poate fi ameliorată prin folosirea unui timeout pentru compilarea documentelor, dar acest lucru poate fi problematic pentru documente foarte mari.

În plus, se folosește un utilizator special pentru execuția programului `pdflatex` astfel încât să fie limitată compromiterea procesului prin codul rulat.

Acest lucru permite în Linux și setarea de limite:

- spre unde poate scrie un utilizator;
- câtă memorie virtuală poate utiliza (folosind `ulimits`);
- cât timp de CPU poate folosi, și altele.

Codul rău nu este în mod necesar maliții deci un utilizator trebuie prezentat cu o eroare în loc să fie interzis.

3.6.3 Altele

Mai sunt alte feluri în care se poate influența sistemul, spre exemplu prin scrierea de fișiere folosind pachetul `newfile`. Așa se poate umple sistemul de fișiere. Și această problemă este rezolvată prin setarea de limite pe utilizator.

3.7 Compilator LaTeX

La momentul de față modulul pentru compilarea LaTeX face parte din aplicația web.

Compilarea documentelor LaTeX este partea cea mai costisitoare din punctul de vederea al procesorului, deci ar trebui extrasă într-un serviciu separat (discutat în secțiunea **??** de la pagina **??**).

3.8 Structură proiect

În baza de date un proiect conține:

- metadate de utilizare (data creației, număr de vizualizări etc);

- autorul principal sub care este listat proiectul;
- setul de colaboratori care pot edita proiectul;
- lista de versiuni (cu toate fișierele în file store);
- locația dosarului de lucru (denumit head) unde este stocată starea curentă a proiectului;
- lista de fișiere sursă;
- și alte câmpuri.

Dosarul de lucru trebuie să existe din motivele listate la secțiunea **??** de la pagina **??**. El poate fi accesat din editor doar de către colaboratori și este făcut public când este făcută o nouă versiune din starea curentă (commit).

Este important să fie enumerate fișierele sursă în baza de date pentru a fi distinse de cele generate de LaTeX. Fișierele generate trebuiesc ignorate la incluziunea în istoric. Singura excepție este însăși fișierul PDF rezultat care este inclus separat.

Fiecare fișier sursă are un cod numeric. Inițial foloseam însăși calea fișierului drept cod (spre exemplu în comunicarea dintre colaboratori la editare), dar acest lucru cauzează probleme la redenumire.

3.9 Versionare

Modul de memorarea a istoricului de editare este inspirat de Git. Mereu se stochează fișierele noi rezultate și nu doar diferențele. Acest lucru duce la folosirea a mai mult spațiu pe disc, dar se câștigă viteză pentru că nu trebuiesc citite toate modificările intermediare pentru a se reconstrui o anumită versiune. În secțiunea **??** de la pagina **??** arăt cum s-ar putea reduce acest lucru.

Ca și în Git, unitatea de bază este setul de modificare a proiectului și nu este folosit un istoric separat pentru fiecare fișier în parte (cum se procedează la SVN).

Când se creează o nouă versiune a proiectului toate fișierele sursă sunt introduse în file store unde primesc și un cod (rezumatul SHA1). Dacă proiectul compilează cu succes, este introdus și PDF-ul rezultat.

Pentru fiecare pagină din PDF este făcută o miniatură. Miniatura dinamică (explicată în secțiunea **??**, pagina **??**) este afișată pe pagina proiectului și în galerie.

Dacă fișierul a fost compilat cu succes, atunci se face și o arhivă ZIP cu conținutul proiectului. Arhiva poate fi folosită pentru exportare și conține și documentul PDF (dar nu și celelalte fișiere generate de `pdflatex`). Am ales formatul ZIP pentru că este printre cele mai recunoscute.

Se poate face duplicare (fork) la oricare versiune a unui proiect, nu doar la versiunea cea mai recentă.

În baza de date o versiune conține:

- metadate specifice;
- lista fișierelor sursă din file store identificate după rezumatul SHA1;
- fișierul PDF din file store (dacă există);
- arhiva ZIP din file store (dacă există);
- lista fișierelor miniatură (file store);
- lista de colaboratori care au contribuit la versiunea curentă;
- lista de modificări efectuate față de versiunea precedentă (pentru a nu fi calculate la fiecare a fișare în pagina web);
- și altele.

3.10 Notificări

Pentru utilizatorii care editează concurent un proiect este foarte important să poată comunica în mod direct. Din acest motiv am implementat notificările în timp real care fac parte din editor. Notificările principale sunt de tip chat, dar sunt și cele generate automat din evenimente specifice (spre exemplu când se face un commit nou al proiectului).

În baza de date notificările sunt stocate doar prin lista de mesaje și interpretarea lor este lăsată pentru editorul de pe partea de client.

În acest mod anumite mesaje pot fi adnotate. Spre exemplu, dacă un utilizator trimite mesajul „Uită-te la conținut.tex:84.“, atunci fraza „conținut.tex:84“ va fi o legătură la linia respectivă (și se deschide fișierul dacă este nevoie).

3.11 File store

File store-ul este o aplatizare a fișierelor din toate proiectele după rezumatul SHA1. Fișierele adăugate în el nu mai pot fi modificate (deoarece n-ar mai corespunde rezumatul). File store-ul dispune de următoarele avantaje.

- Dacă la o versiune nouă nu se modifică un fișier (sau se revine la o versiune mai vechi) nu trebuie stocat din nou.
- Se reduce costul duplicării proiectelor (forking) la aproape zero pentru că nu este nevoie să fie creat niciun fișier nou la duplicare și până în momentul separării au un istoric comun.
- Oferă un nivel sporit de siguranță a datelor pentru că se poate verifica în viitor corespondența dintre numele din file store și rezumatul SHA1 recalculat.

Toate fișierele sunt stocate într-un dosar, dar deoarece nu toate sistemele de fișiere suportă un număr mare de fișiere într-un singur dosar, se realizează o grupare după primele trei caractere din cod.

Deci `45ad3b07...` este stocat în `$file_store/45a/d3b07...`.

3.12 Comunicare

Comunicarea dintre editorul de pe partea client și serverul se face printr-un RPC prin protocolul WebSocket și colaborarea asupra unui fișier se face folosind ShareJS (care la fel folosește WebSocket sau Comet).

3.12.1 WebSocket

Când un utilizator se conectează la pagina editorului se stabilește o conexiune WebSocket (descris la secțiunea **??**, pagina **??**) cu controlorul de pe server.

Dacă există mai mulți utilizatori conectați simultan aceștia sunt notificați de schimbările celorlalți. Dacă sunt mai mulți care editează un fișier, editarea lor este administrată prin ShareJS. În plus, ei pot comunica direct prin notificări (vezi secțiunea **??** la pagina **??**).

Pe partea de server pentru comunicarea WebSocket folosesc pachetul Node numit `ws`. El are avantajul că imită comunicarea JavaScript de pe client.

3.12.2 ShareJS

ShareJS (descris la secțiunea **??**, pagina **??**) este folosit pentru colaborarea pentru editarea unui fișier text în parte. Dacă mai mulți utilizatori editează fișiere diferite nu există probleme de sincronizare între ei (dar tastele tot sunt transmise în timp real către server).

Problemele de consistentă apar când mai mulți utilizatori editează un fișier. ShareJS asigură că documentul va ajunge la aceeași stare indiferent de latența dintre utilizatori.

3.13 Editor

Editorul (afișat în figura **??**) este compus ca o aplicație care folosește întreaga pagină. Deoarece necesită un ecran mare, editorul nu este suportat și pe platforme mobile.

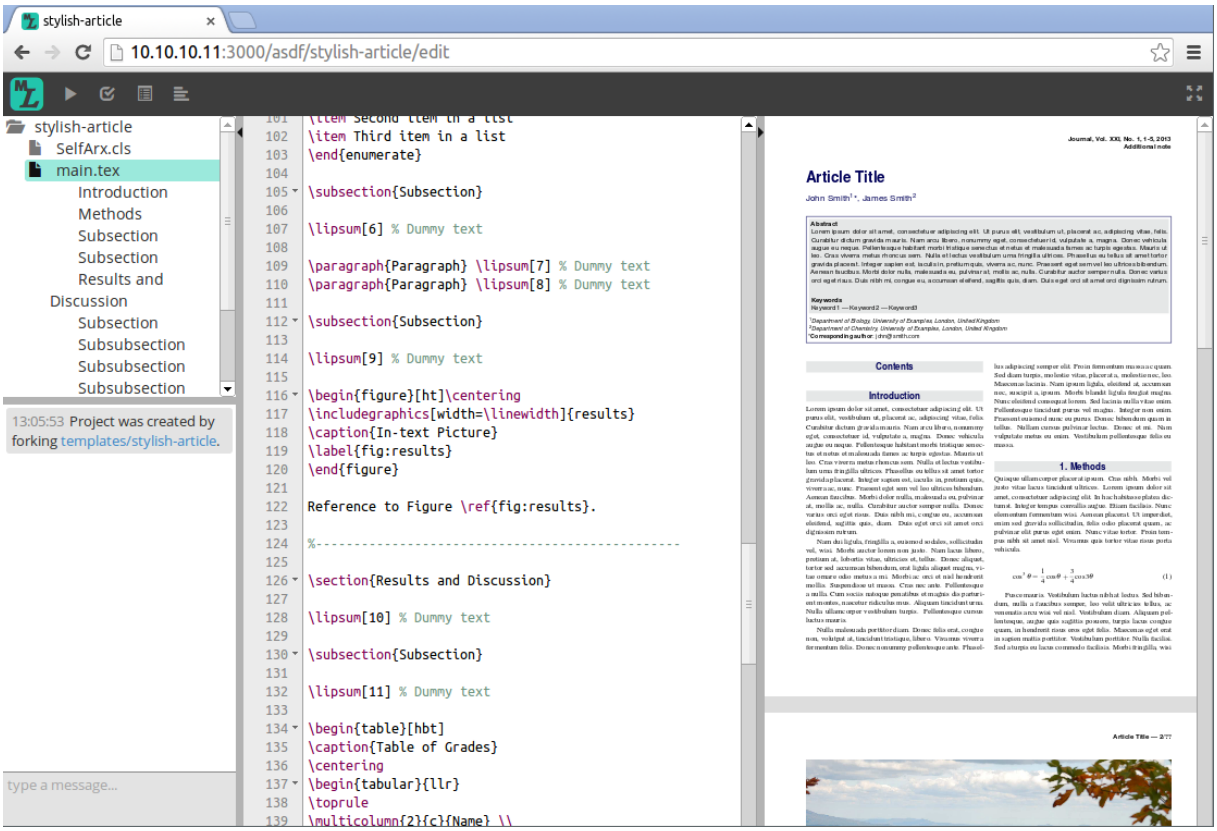

Figura 3.4: Editorul

El are și funcționalitatea de a fi folosit pe tot ecranul (fullscreen).

Editorul este compus din patru părți: meniul, panoul de proiect, panoul de editare și panoul de previzualizare.

Cele trei panouri principale sunt separate de marcaje de control. Panoul din stânga (de proiect) poate fi ascuns în stânga prin marcajul de control și panoul din dreapta (de previzualizare) poate fi ascuns în dreapta. Panoul de editare este singurul care nu poate fi minificat.

Editorul are mai multe setări care pot fi controlate fie prin meniu fie prin controale. Setările sunt sincronizate în timp direct cu serverul și prin urmare vor fi disponibile de fiecare dată când este redeschis documentul.

Inițial plănuiam să fie două nivele de setări: cele individuale și cele de proiecte care să aibă prioritate. Însă au fost implementate doar cele individuale.

Din meniu poate fi și compilat documentul LaTeX.

3.13.1 Meniu

În meniu sunt organizate toate acțiunile care au efect asupra întregului proiect. Pentru a fi mai compact, ele sunt reprezentate doar prin iconițe de tip font. Iconițele afișează meniul text la hover.

Funcționalitățile implementate în meniu sunt de:

- compilare a documentului LaTeX;
- crearea unei versiuni noi (commit);
- descărcarea proiectului în starea curentă (din pagina proiectului se poate descărca doar cea mai nouă versiune);
- vizualizarea fișierului de logare de la compilarea documentului (pentru depanare);
- și modificarea setărilor de editor.

3.13.2 Proiect

Panoul de proiect are ca rol să reprezinte starea curentă a proiectului dintr-un punct înalt de vedere. În momentul de față el este compus din arborele de fișiere și lista de notificări.

Arbore de fișiere

Arborele de fișiere reprezintă lista curentă de fișiere a proiectului și ceea ce se poate realiza cu fiecare fișier. Acțiunile asupra fișierelor sunt afișate doar când este selectat un fișier sau se efectuează operația de hover asupra lui.

Proiectul este reprezentat ca rădăcina fișierelor. De acolo pot fi controlate și acțiunile care pot fi efectuate asupra lui.

Acțiunile disponibile sunt afișate doar când sunt necesare pentru a nu confuza.

Notificări

Panoul de notificări conține toate notificările proiectului care este editat în mod curent. Tipurile de notificări sunt discutate la secțiunea **??** de la pagina **??**.

3.13.3 Editare text

Panoul de editare de text este cel care conține editorul Ace (descris la secțiunea **??**, pagina **??**). Acesta este configurat special pentru editarea de fișiere de tip LaTeX cu tot cu colorarea sintactică. Alte tipuri de setări ale editorului pot fi accesate din meniu.

3.13.4 Previzualizare

Panoul de previzualizare este cel care conține paginile randate prin PDF.js. Paginile sunt recreate de fiecare dată când este compilat documentul LaTeX. Deoarece paginile sunt create ca imagini în etichete `canvas` ele trebuiesc recreate când se redimensionează mărimea panoului pentru că altfel n-ar fi la fel de clare.

3.14 Șabloane

Proiectul conține și o listă de șabloane precompilate cu rolul de:

- a demonstra imediat posibilitățile de documente care pot fi create cu LaTeX;
- a testa rapid funcționalitățile și modul de lucru oferit de Multilatex;
- și de a constitui un punct de plecare pentru diferitele tipuri de formate suportate (pentru a fi duplicate ușor).

Ele au fost alcătuite din proiecte LaTeX libere de pe internet cu scopul de a acoperi o arie cât mai largă a tipurilor de documente care pot fi editate cu LaTeX și anume: eseuri, articole, cărți, raporturi și prezentări Beamer.

Inițial șabloanele făceam parte din proiect, dar odată cu mutarea dependințelor Bower în afară am mutat și șabloanele ca un modul separat care este inclus în proiect prin funcționalitatea submodule din Git. Astfel se pot dezvolta în paralel și pot fi folosite și pentru alte proiecte.

3.15 Colaborare

În Multilatex nu există branching ca în Git sau alte sisteme de versionare (pentru că ar complica sistemul) și din acest motiv duplicarea proiectelor este motoda recomandată pentru a încerca lucruri noi în paralel.

Pentru a încerca o editare care modifică în mod radical documentul un utilizator în poate duplica proiectul și poate să facă modificările cu mai multe commit-uri fără a intra în conflicte cu ceilalți colaboratori.

Dacă modificările sunt considerate bune ele pot fi incluse în proiectul principal.

Chiar și autorul proiectului îl poate duplica pentru a încerca lucruri noi.

Capitolul 4

Îmbunătățiri

În acest capitol se discută o serie de îmbunătățiri care pot fi aduse proiectului. Sunt multe feluri în care poate fi făcut mai bun, dar concentrarea o să fie pe cele care pot fi aduse proiectului în starea sa curentă.

4.1 Compilator LaTeX

Datorită faptului că compilarea documentelor LaTeX este partea cea mai costisitoare, ea ar trebuie separată de aplicația web.

Metoda cea mai bună ar fi crearea unui serviciu web special pentru compilarea documentelor LaTeX care să fie scalabil.

Acțiunile unui worker în parte nu ar fi greu de implementat deoarece nu ar trebui să fie decât un wrapper pentru utilizarea programului `pdflatex` ca în sistemul propus.

4.2 File store

Există cel puțin două modalități prin care poate fi îmbunătățită metoda curentă: se pot comprima fișierele și se pot refolosi fișierele din file store în dosarul de lucru.

4.2.1 Comprimare

Deoarece cele mai multe fișiere sunt de tip text, ele pot fi comprimate ușor. În acest caz pot fi folosite sisteme de fișiere cu suport pentru comprimarea fișierelor sau fiecare fișier să fie comprimat la scriere și decomprimat la citire.

Însă aceste soluții nu ar fi chiar cele mai bune. O comprimare mai eficientă ar ține cont de faptul că există diferențe mici între multe fișiere, spre exemplu, un fișier pentru care se modifică la fiecare versiune doar câteva linii de text. Sistemele de versionare mai vechi rezolvau această problemă prin memorarea doar a diferențelor aduse în fiecare versiune (RCS[**?**], CVS etc).

O altă soluție ar fi folosirea unui sistem de fișiere cu comprimare. Formatul de compresie ar folosi dicționare de mărime fixă care, posibil, ar fi mai eficient de încărcat în memorie și decomprimat decât reconstruirea unei versiuni specifice prin aplicarea diferențelor dintre versiuni.

Sistemele de fișiere care comprimă bine de obicei sunt read-only (SquashFS, cramfs etc).

O variantă în acest caz ar fi scrierea unui sistem de fișiere virtual (folosind FUSE pentru Linux) care să arhiveze doar fișierele text și accesul la restul să fie furnizat direct de disc prin folosirea unui union mount (spre exemplu cu UnionFS).

4.2.2 Legături

Deoarece unele fișiere din dosarul de lucru se schimbă rar (mai ales cele de dimensiuni mai mari cum ar fi resursele), o optimizare posibilă ar fi să se folosească hard link-uri către fișierele din file store. Bineînțeles, legăturile trebuiesc înlocuite dacă se editează fișierul pentru a nu fi alterate fișierele statice din file store.

4.2.3 Altele

Alte modalități ar fi să fie șterse periodic fișierele generate care nu sunt accesate de multe ori. La secțiunea **??** de la pagina **??** explic că documentul PDF și arhiva ZIP sunt create la fiecare commit. Ele ar putea fi șterse și recreate dacă este din nou nevoie de ele.

4.3 Înlocuire ShareJS

Deși ShareJS funcționează corect există probleme de performanță:

- Pentru Multilatex fișierele editate sunt stocate pe disc în dosarul de lucru al proiectului, dar ShareJS folosește o bază de date deci se face transferul inutil între baza de date și fișierul normal.

- ShareJS memorează fișierul pe client ca un singur șir de caractere ceea ce este încet deoarece JavaScript nu poate modifica șirurile de caractere și trebuie alocată și dealocată multă memorie pentru fiecare schimbare a documentului.

ShareJS ar putea fi înlocuit cu biblioteci ca OT.js[**?**] care sunt mai simple și au mai puține funcționalități.

Concluzii

Implementarea acestei aplicații web care mută un program tradițional static în cloud se dovedește a fi una bună în contextul creșterii în utilizare a cloud computing.

Peste modelul tradițional o astfel de aplicație aduce avantaje de:

- securitate, pentru că date sunt sincronizate în timp real cu serverul și nu este nevoie de nicio salvare explicită;
- productivitate, pentru că se poate colabora la editarea unui document și poate fi accesat de oriunde;
- performanță, pentru că nu necesită instalarea unei distribuții de LaTeX și nu se consumă CPU compilând documentele;
- și de ușurință în experimentare deoarece sunt salvate toate versiunile și se poate duplica un proiect pentru a încerca lucruri noi.

Bibliografie

[1] mirrors.ctan.org/systems/knuth/dist/tex/tex.web

[2] brokestream.com/tex.pdf

[3] Andrew Marc Greene. BASIX — An Interpreter Written in TEX. TUGboat, Volume 11 (1990). No. 3

[4] Kent Beck. Extreme Programming: A Humanistic Discipline of Software Development. Fundamental Approaches to Software Engineering. Proceedings, Volume 1. Springer. 1998

[5] C. A. Ellis, S. J. Gibbs. Concurrency Control in Groupware Systems. ACM SIGMOD Record 18. 1989

[6] Walter F. Tichy. RCS—a system for version control. Journal Software—Practice & Experience Volume 15 Issue 7. 1985

[7] Glenn E. Krasner, Stephen T. Pope. A cookbook for using the model–view–controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, Volume 1, Issue 3. 1988

[8] I. Fette, A. Melnikov. The WebSocket Protocol. RFC 6455. 2011

[9] thepasswordproject.com/leaked_password_lists_and_dictionaries

[10] manuels.github.io/texlive.js

[11] gs.statcounter.com

[12] ace.c9.io

[13] backbonejs.org

[14] en.wikibooks.org/wiki/LaTeX/Installation#Distributions

[15] mozilla.github.io/pdf.js

[16] upload.wikimedia.org/wikipedia/commons/5/58/Basicot.png

[17] github.com/Operational-Transformation/ot.js

[18] github.com/sharelatex/sharelatex

[19] nodejs.org

[20] developers.google.com/v8/intro

[21] www.mongodb.org

[22] www.vagrantup.com

[23] upstart.ubuntu.com

[24] expressjs.com

[25] jade-lang.com

[26] learnboost.github.io/stylus

[27] bower.io

[28] github.com/share/ShareJS/tree/0.6

[29] gruntjs.com

[30] browserify.org

[31] gulpjs.com

[32] github.com/paul-nechifor/multilatex (proiect)
github.com/paul-nechifor/multilatex-dissertation (lucrare)