UNIVERSITATEA „AUREL VLAICU” DIN ARAD
FACULTATEA DE ȘTIINȚE EXACTE
DOMENIUL: INFORMATICĂ
PROGRAMUL DE STUDIU: MASTER -
INFORMATICĂ APLICATĂ ÎN ȘTIINȚE,
TEHNOLOGIE ȘI ECONOMIE
FORMA DE ÎNVĂȚĂMÂNT: IF

LUCRARE DE DISERTAȚIE

ÎNDRUMĂTOR ȘTIINȚIFIC
Prof. univ. dr. Ioan Dziţac

ABSOLVENT
Șucan M. Mihai

ARAD
2013

UNIVERSITATEA „AUREL VLAICU” DIN ARAD
FACULTATEA DE ȘTIINȚE EXACTE
DOMENIUL: INFORMATICĂ
PROGRAMUL DE STUDIU: MASTER -
INFORMATICĂ APLICATĂ ÎN ȘTIINȚE,
TEHNOLOGIE ȘI ECONOMIE
FORMA DE ÎNVĂȚĂMÂNT: IF

METODE AVANSATE ÎN PROIECTAREA UNUI WEBSITE UNIVERSITAR
Aplicație practică: www.univagora.ro

ÎNDRUMĂTOR ȘTIINȚIFIC
Prof. univ. dr. Ioan Dziţac

ABSOLVENT
Șucan M. Mihai

ARAD
2013

Cuprins

Introducere

Lucrarea de față prezintă metode avansate de realizare a unei aplicații web, având în vedere proiectul realizat pentru Universitatea Agora din municipiul Oradea. Scopul proiectului a fost realizarea unui web site modern, după standarde și tehnici actuale, ușor de actualizat și de menținut.

Împreună cu Marius Șucan am realizat următoarele:

- o interfață nouă pentru siteul Universității Agora;
- un plan de structură;
- un proiect Python bazat pe pachete *open source*, fără a reinventa roata.

Marius a realizat interfața aplicației, iar eu m-am ocupat de programare, de partea tehnică a acestui proiect. Domnul profesor coordonator Ioan Dziţac s-a ocupat de completarea siteului cu conținut.

Aplicația web este flexibilă, poate fi instalată pe orice server web care implementează tehnologii și standarde deschise:

- Orice sistem de operare: Linux, Mac OS X sau Windows;
- Orice server web cu suport WSGI: nginx, uwsgi[1], lighttpd[2], Apache și altele.
- Bază de date SQLite, PostgreSQL sau MySQL la alegere. Dacă se dorește se poate dezvolta suport și pentru alte tipuri de baze de date.

Codul sursă al aplicației web este distribuit și găzduit într-un depozit Git pregătit pentru colaborarea între mai mulți programatori. Aplicația poate fi dezvoltată în colaborare de o echipă fără nici o problemă tehnică.

Scopul proiectul a fost de a realiza un site nou, cu o interfață mai frumoasă, mai ușor de navigat și de administrat, cu conținut bilingv. Noul site este mai bine structurat și mai flexibil - poate fi actualizat și îmbunătățit în orice moment.

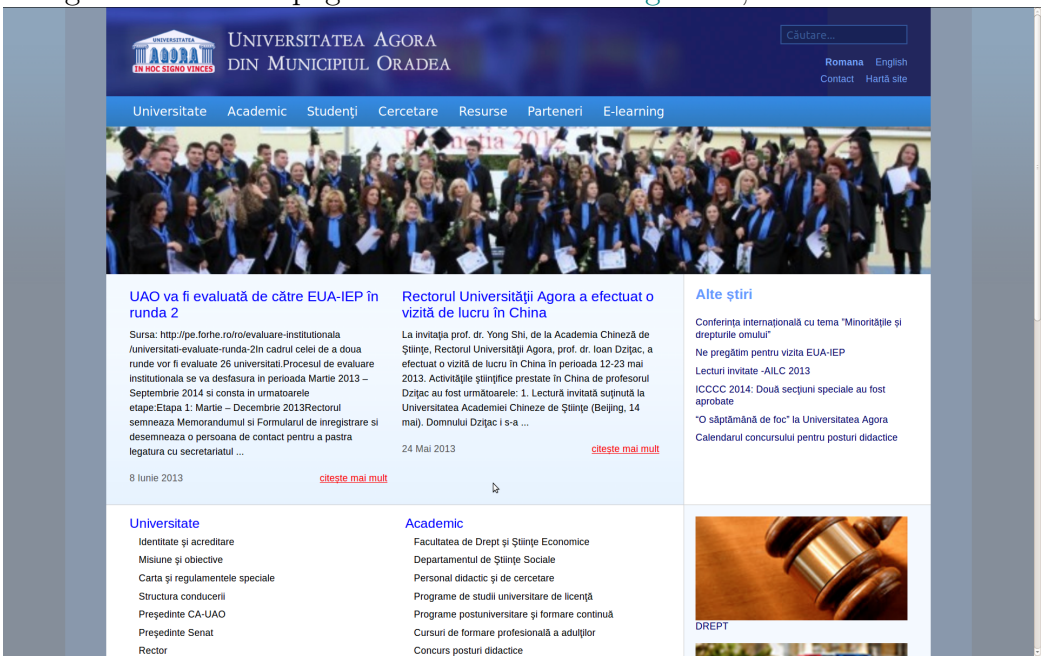
[1] https://projects.unbit.it/uwsgi/
[2] http://lighttpd.net

Capitolul 1

Prezentarea siteului

În acest capitol includ o prezentare generală a siteul: structura acestuia, funcționalități pentru utilizatori și administratori, și compatibilitatea siteului cu navigatoarele web moderne.

Figura 1.1: Prima pagină a noului site univagora.ro, în limba română.



1.1 Structura siteului pentru utilizatori

Siteul Universității Agora este structurat în câteva secțiuni importante:

Universitate este secțiunea unde vizitatorii pot afla informații despre Universitatea Agora, istoric, noutăți, poze și alte detalii.

Academic este secțiunea unde se pot găsi informații despre programele de studii oferite, despre Facultatea de Drept și Științe Economice, despre personalul didactic și de cercetare, programe postuniversitare, și concursuri pentru posturi didactice.

Studenți este locul unde studenții universității găsesc date despre structura anului universitar în curs, despre admitere, orare, burse, taxe, examene și alte informații la fel de importante.

Cercetare este secțiunea unde persoanele interesate se pot informa despre proiectele de cercetare ce sunt în derulare, despre conferințe și centrul de cercetare al universității.

Resurse este locul unde vizitatorii siteului găsesc informații despre biblioteci, locuri de muncă, editura universității și platforma de învățământ la distanță Moodle.

Parteneri este secțiunea dedicată prezentării parteneriatelor naționale și internaționale cu alte universități și instituții.

1.2 Funcționalități pentru utilizatori

Utilizatorii siteului se pot bucura de următoarele funcționalități ale siteului:

- Conținut bilingv,
- Interfață adaptată pentru diferite ecrane și pentru imprimare,
- Blog[1],
- Galerie de poze[2],
- Formular contact[3],

[1] http://univagora.ro/ro/universitate/noutati/
[2] http://univagora.ro/ro/universitate/foto/
[3] http://univagora.ro/ro/contact/

- Hartă site[4],
- Căutare[5].

Pe prima pagină a siteului se poate vedea o diaporamă cu ultimele poze adăugate în site. Sub diaporamă sunt afișate legături către cele mai importante pagini din site, și o listă cu ultimele articole din blogul „noutăți".

Blogul „noutăți" permite filtrarea articolelor după etichete, *tags*, și categorii. Dacă utilizatorii doresc, aceștia se pot abona la știri pentru a fi notificați când apar articole noi cu ajutorul fluxului de știri în format Atom.

Galeria de poze este împărțită pe albume. Fiecare album poate să aibă o descriere și numeroase poze. Utilizatorii pot alege să filtreze pozele după etichete.

Formularul de contact permite alegerea destinației și include protecția anti-spam a serviciului online recaptcha.net.

Motorul de căutare folosește o librărie Python numită Whoosh. Aceasta permite căutarea cu operatori booleeni - de exemplu AND și OR. Căutarea în site este foarte rapidă și flexibilă: sunt indexate toate paginile și fiecare tip de pagină are afișare specifică în lista de rezultate.

Harta siteului este generată separat pentru limbile română și engleză.

Fiecare pagină are opțiunea de mărire și micșorare a textului pentru a îmbunătăți accesibilitatea siteului. În plus, utilizatorii pot alege să tipărească orice pagină.

1.3 Funcționalități pentru administratori

Pentru gestionarea conținutului aplicația web include o secțiune de administrare unde proprietarii siteului pot face modificările dorite.

Administratorii pot intra pe `http://www.univagora.ro/admin` pentru a edita conținutul siteului. Aceștia au nevoie de conturi pentru a se putea autentifica.

Secțiunea de administrare a siteului include funcții specializate pentru fiecare tip de conținut: se pot adăuga/edita/șterge pagini, albume de poze, poze din albume, articole pe blog, etichete, etc.

La fiecare câmp de text, orice titlu de articol, de poză poate fi scris în engleză și în română. Siteul poate fi tradus în cele două limbi într-un mod

[4] `http://univagora.ro/ro/harta-site/`
[5] `http://univagora.ro/ro/cautare/`

flexibil: se pot adăuga pagini și articole pe blog în ambele limbi sau, la alegere, doar într-o limbă.

Editarea de pagini și articole în blog include un editor HTML simplu de folosit. Editorul de pagini include și opțiuni pentru adăugarea de înregistrări video, poze, fișiere de descărcat (atașamente) și legături spre alte pagini. Fiecare pagină poate fi afișată în meniuri sau ascunsă, poate fi publicată sau nu - administratorul poate adăuga pagini de tip „ciornă". Paginile pot fi ordonate într-o structură de tip arbore cu ierarhie nelimitată.

Noțiunea de pagini „ciornă" este implementată și la albumele de poze și la articolele de pe blog: administratorii pot adăuga albume de poze și articole pe blog care nu vor fi publice până când se dorește.

Pentru adăugarea de poze administratorii pot alege să încarce un fișier ZIP cu toate pozele, într-un singur pas. Toate pozele încărcate sunt redimensionate automat în funcție de designul siteului.

1.4 Compatibilitatea siteului

Siteul este compatibil cu:

- Mozilla Firefox[6],
- Google Chrome[7],
- Apple Safari[8],
- și Microsoft Internet Explorer[9].

Siteul poate fi folosit și pe navigatoare web ce nu au suport pentru JavaScript sau CSS. Siteul poate fi navigat și fără afișarea de imagini - cu ajutorul sintetizatoarelor vocale sau cu a sistemelor Braille.

[6] http://www.mozilla.com
[7] http://www.google.com/chrome
[8] http://www.apple.com/safari
[9] http://www.microsoft.com/windows/internet-explorer/

Capitolul 2

Modul de funcționare al siteului

Siteul Universității Agora este o aplicație web implementată în limbajul de programare Python, construită cu Django. În acest capitol voi explica modul de funcționare al sitelui, cum este folosit Django, de ce a fost ales, tehnologiile care stau la baza aplicației web, cerințele tehnice pentru server și, nu în ultimul rând, voi explica fiecare pagină din site.

2.1 Tehnologiile server-side și client-side folosite de aplicație

Tehnologiile folosite de aplicația web sunt următoarele:

- Git este folosit pentru dezvoltarea și distribuirea aplicației web. În acest mod se pot face modificări la site mult mai ușor, mai rapid și mai sigur - istoricul modificărilor este menținut în depozitul git.
- Python este limbajul de programare în care este scris tot codul aplicației. Cu ajutorul acestuia fiecare pagină HTML este generată dinamic.
- Django este cadrul de dezvoltare al aplicației. Acesta oferă un model formal de structurare a aplicației web, de scriere a codului și numeroase funcții specifice aplicațiilor web.
- Bază de date la alegere: SQLite, MySQL sau PostgreSQL. Toată structura siteului este salvată în baza de date: paginile, articolele de pe blog, lista de poze și albume, etichetele, etc.

- Whoosh este folosit pentru indexarea și căutarea paginilor din site.
- HTML este folosit pentru paginile siteului.
- CSS este folosit pentru designul siteului, fără tabele pentru *layout*.
- JavaScript este folosit pentru secțiunile interactive din site.
- Fluxurile Atom sunt folosite pentru paginile blogului „noutăți”. Acest format XML permite utilizatorilor să se aboneze la blog, să fie informați oricând sunt adăugate alte articole.
- Sitemaps XML este un format standard în care lista de pagini din site este publicată pe server pentru motoarele de căutare gen Google. Acestea pot să verifice ultimele modificări din site mai ușor și mai rapid [1].

2.2 Cerințele tehnice pentru server

Serverul are nevoie de următoarele aplicații:

- Python 2.7.
- Un server web cu suport WSGI, de exemplu nginx sau Apache - se recomandă prima variantă.
- O bază de date PostgreSQL sau MySQL - se recomandă a doua variantă.
- Pachetele Python `virtualenv` [1] și `pip` [2].
- Git pentru descărcarea codului sursă al siteului.

Se preferă server cu un sistem de operare Linux datorită compatibilității mai bune a software-ului folosit.

[1] `https://pypi.python.org/pypi/virtualenv`
[2] `https://pypi.python.org/pypi/pip`

2.3 Privire pe ansamblu

Serverul web este responsabil cu procesarea cererii de accesare a unei pagini vizitate de utilizator. Pașii urmați de server sunt:

- Cererea din navigatorul web al utilizatorului este trimisă prin protocolul HTTP la server.
- Dacă utilizatorul accesează un fișier static, de exemplu o imagine, atunci fișierul este trimis direct la client.
- Dacă se dorește afișarea unei pagini dinamice, atunci serverul web transmite cererea mai departe la aplicația Django cu ajutorul interfeței WSGI. La fel ca un server web, aplicația Django este un proces ce rulează constant pe server, care așteaptă cereri de afișare pagini.
- Aplicația Django verifică adresa URL și restul de antete HTTP și execută scripturile asociate paginii accesate de utilizator.
- Pagina generată de Django este trimisă la serverul web care apoi o trimite la navigatorul web al utilizatorului.

Django este un pachet Python, iar siteul este un proiect Django. Orice proiect Django are în componența sa reguli de asociere a adreselor acceptate de aplicația web - de exemplu `/despre-noi` sau `/contact`. Fiecare adresă acceptată este asociată cu funcții Python din aplicația web.

Majoritatea aplicațiilor web Django se bazează și pe alte proiecte cu sursă liberă, *open source*. Pe lângă pachetul Django care poate fi instalat cu `pip` pe un sistem unde este Python, există și alte pachete pentru Django în comunitatea proiectului. Alți utilizatori Django își publică pe Internet proiectele lor, sub licențe libere.

În secțiunile următoare voi prezenta proiectul Django mai detaliat, pachetele folosite de site, și structura de foldere și fișiere din aplicația web.

2.3.1 Proiectul Django

Majoritatea siteurilor sunt făcute cu PHP și MySQL[3], motiv pentru care sunt foarte bine cunoscute problemele de securitate asociate limbajului PHP[4] -

[3] conform Wikipedia: https://en.wikipedia.org/wiki/PHP#Use
[4] https://en.wikipedia.org/wiki/PHP#Security

foarte des siteuri care folosesc PHP sunt subiectul atacurilor. Pentru Universitatea Agora mi-am propus să realizez un proiect scalabil care să se ridice la nivelul celor mai performante aplicații web din lume.

Django este un proiect care are versiuni lansate la intervale bine definite, cu documentație foarte bună și informații pertinente despre fiecare versiune - cum se poate migra de la o versiune la alta [6]. În plus, proiectul este orientat către programatorii marilor companii care își doresc în primul rând calitate, stabilitate și securitate.

Printre companiile care folosesc Django amintesc Mozilla[5], OpenStack[6], Disqus[7] și Instagram[8] - siteuri foarte populare la nivel mondial.

Cele mai importante concepte în proiectele Django sunt: *models*, *views* și *templates*. Modelele definesc tabelele cu câmpurile lor și relațiile între ele. *Views* sunt clasele/funcțiile care controlează navigarea utilizatorului în aplicația web. Șabloanele sunt bucățile de cod HTML ce preiau datele din *views* pentru afișare - interfața aplicației.

Exemplu de model:

```
from django.db import models
from django.utils.translation import ugettext_lazy as _

class ContactGroup(models.Model):
    title = models.CharField(_('title'), \
                             max_length=150)

    class Meta:
        verbose_name = _('contacts group')
        verbose_name_plural = _('groups of contacts')

    def __unicode__(self):
        return self.title

class ContactRecipient(models.Model):
    group = models.ForeignKey(ContactGroup, \
                              verbose_name=_('group'))
    display_name = models.CharField(_('name'), \
                                    max_length=150)
    email = models.EmailField(_('email'), \
```

[5] http://mozilla.com
[6] http://openstack.org
[7] http://disqus.com
[8] http://instagram.com

```
                                    max_length=100)

    class Meta:
        verbose_name = _('recipient')
        verbose_name_plural = _('recipients')
        unique_together = (('group', 'email'), \
                           ('group', 'display_name'))

    def __unicode__(self):
        return self.display_name
```

Modelul de mai sus este preluat din `apps/robo/contact/models.py`. Aplicația pentru Contact folosește cele două modele: `ContactRecipient` și `ContactGroup` pentru pagina contact. Se poate vedea că fiecare destinatar are numele, adresa de email și grupa în care se găsește. Fiecare grupă are doar un titlu. Aceste două tabele sunt adăugate automat în baza de date a aplicației web.

Tabelele se definesc doar sub forma de modele - clase Python care moștenesc clase definite în proiectul Django. Nu este nevoie de nici un altfel de configurare: la instalarea aplicației web toate aceste fișiere Python sunt folosite pentru a genera baza de date. Django se ocupă cu diferențele dintre MySQL, PostgreSQL și SQLite. La rularea siteului aceste modele sunt folosite și pentru citire, modificare, adăugare și ștergere de date din tabele. Programatorul nu are nevoie să lucreze cu baza de date direct - totul este abstractizat de aceste modele.

O pagină sau un *view* în Django arată în felul următor:

```
from django.shortcuts import render_to_response, \
                             get_object_or_404
import blog

def article(request, article_id):
    object = get_object_or_404(BlogArticle, \
                               pk=article_id)
    return render_to_response('blog/article.html', \
                              {'article': object})
```

Acest *view* afișează un articol din blog pe baza ID-ului, cu ajutorul unui șablon. Șablonul `blog/article.html` poate fi următorul:

```
<article>
<h1>{{ article.title }}</h1>
```

```
<p>Publicat in {{ article.date|date:"DATE_FORMAT" }}</p>
{{ article.html_content }}
</article>
```

Orice *view* poate fi asociat la o adresă URL în fișierul `urls.py` al proiectului Django, în lista de `urlpatterns`. Formatul este următorul:

```
(expresie regulata, functie Python[, dictionar optional])
```

Când este o cerere HTTP de la un vizitator Django trece prin lista de `urlpatterns` și caută să vadă dacă adresa cerută există. Prima expresie regulată care se potrivește cu URL-ul paginii dorite este folosită - Django execută funcția Python asociată. Exemplu de asociere:

```
from django.conf.urls.defaults import patterns, url
urlpatterns = patterns('',
  url(r'^blog/$', 'blog.views.index'),
  url(r'^blog/(?P<id>\d+)/$', 'blog.views.article'),
)
```

Conceptele de mai sus sunt suficiente pentru realizarea de aplicații web cu Django: tabele în baza de date, pagini, șabloane și adrese URL. Bineînțeles, Django include mult mai multe opțiuni și capabilități.

Structura *model-view-template* din Django este foarte similară cu structura de aplicații software mai bine cunoscută sub numele de *model-view-controller*. Cei de la Django au ales să schimbe puțin terminologia.

2.3.2 Pachetele folosite de site

Siteul Universității Agora folosește câteva pachete Python. Dependințele sunt listate în fișierul `deploy/requirements.txt` - acest fișier este folosit la instalarea aplicației web pe server. Pachetele Python:

Django este cadrul aplicației web. [9]

PIL (Python Image Library) [10] este folosit pentru procesarea imaginilor adăugate în galeria de poze a siteului.

Whoosh este un motor de indexare și de căutare documente. [11]

[9] http://djangoproject.com
[10] http://www.pythonware.com/products/pil/
[11] https://bitbucket.org/mchaput/whoosh/

Paste este un pachet de unelte pentru aplicații WSGI folosit de site doar în timpul testării și a depanării problemelor. [12]

Pe lângă pachetele listate mai sus siteul folosește și următoarele pachete specifice Django:

django-cms este un sistem de gestiune a conținutului. [13]

django-haystack este un pachet care integrează Whoosh și alte motoare de căutare cu Django. [14]

django-filer este un gestionar de fișiere. [15]

django-blog-zinnia este o aplicație web blog. [16]

imagestore este un proiect pentru gestionarea de albume și poze. [17]

sorl-thumbnail este un pachet pentru manipulare de imagini într-un mod cât mai simplu. [18]

La pachetele de blog și de galerie poze am făcut următoarele modificări:

- am adăugat suport pentru mai multe limbi;
- integrare proprie cu `django-cms` și cu `django-haystack`;
- integrare cu Django sitemaps;
- modificări la fișierele `urls.py` pentru a se potrivi cu scopul siteului;
- modificări la paginile de administrare.

La instalarea siteului pe server se rulează scriptul `deloy/deploy_new.sh`. Acesta crează un mediu Python în care sunt instalate de pe Internet toate pachetele de care depinde aplicația web a Universității Agora. Pentru o execuție cu succes serverul are nevoie de `virtualenv`, `pip` și de Python.

În cele ce urmează voi prezenta principalele pachete folosite de site.

---

[12] http://pythonpaste.org
[13] http://django-cms.org
[14] http://haystacksearch.org
[15] https://github.com/stefanfoulis/django-filer
[16] http://django-blog-zinnia.com
[17] https://github.com/hovel/imagestore
[18] https://github.com/sorl/sorl-thumbnail

2.4 Structura siteului pe server, fișierele folosite

Pentru a fi mai ușor de înțeles unde se găsesc fișierele și codul sursă asociat cu diferitele funcționalități din aplicația web includ mai jos lista de fișiere și foldere cu explicații.

- `apps/`
  Acest folder include pachete Django care nu provin din mediul Python făcut la instalare, `pyenv/`.
  - `robo/`
    Acest folder include pachete Django făcute de mine pentru siteul Universității Agora.
    * `blog/`
      Aplicația Blog a siteului. Aici se găsesc câteva module Python care schimbă funcționalități din `django-blog-zinnia`.
      · `locale/`
        Traduceri în limba română pentru textele folosite de blog.
      · `admin.py`
        Modul Python pentru secțiunile de administrare a articolelor din blog.
      · `cms_app.py`
        Script pentru integrarea blogului cu `django-cms`.
      · `menu.py`
        Script pentru meniurile din site la paginile blogului.
      · `search_indexes.py`
        Script care definește câmpurile de indexare a articolelor din blog - un model `haystack.indexes.SearchIndex` care preia date din modelul `zinnia.models.Entry`.
      · `translation.py`
        Script care definește câmpurile din modelul `zinnia.models.Entry` care pot fi traduse în diferite limbi.
      · `urls.py`
        Script care definește adresele cunoscute de aplicația blog.
    * `contact/`
      Aplicația pentru paginile de tip Contact. Acest folder are structură similară cu folderul descris mai sus. Trebuie menționat că formularul de contact este un `CMSPlugin` care poate fi folosit în orice pagină.

    * `gallery/`
      Aplicația pentru galeria de poze. Aici se găsesc module Python care schimbă funcționalități din `imagestore`: `admin.py`, `cms_app.py`, `menu.py`, `search_indexes.py`, `sitemaps.py`, `translation.py` și `urls.py`.
    * `utils/`
      Funcții utile folosite prin toată aplicația web care nu aparțin unei componente specifice. Aici se găsesc bucăți mici de cod pentru diferite scopuri: *middleware, sitemaps, search indexes, context processors, template tags,* etc.

- `deploy/`
  Acest folder include scripturi pentru instalarea siteului pe server.

  – `wsgi/app.wsgi`
    Scriptul Python care pornește aplicația web. Acest fișier este executat de serverul web - punctul de comunicare dintre server și aplicație. Comunicarea se face cu ajutorul interfeței WSGI.
  – `config.py.dist`
    Fișier de configurare a modului de instalare al aplicației web.
  – `deploy_new.sh`
    Script cu care se poate instala mediul Python necesar pentru aplicația web.
  – `env.py`
    Script Python ce este executat de aplicația web la pornire. Acesta activează sau nu modul de depanare și modifică *environment variables* precum `PATH` și `PYTHON_PATH`.
  – `requirements.txt`
    Lista de pachete Python care se instalează la rularea scriptului `deploy_new.sh` - lista de dependințe.

- `public/`
  Acest folder conține fișiere și foldere statice. Acestea sunt trimise la client direct de către serverul web, fără să fie implicată aplicația Django.

- `site_project/`
  Acest folder conține fișierele specifice proiectului / siteului.

  – `locale/`
    Traduceri în limbile engleză și în română pentru toate textele care apar în interfața siteului.

  - `templates/`
    Fișiere HTML, șabloane Django, care sunt folosite în diferitele secțiuni ale sitelui - blog, galerie de poze, prima pagină, contact, căutare, hartă site, etc.
  - `settings.py`
    Setările siteului.
  - `settings_local.py.dist`
    Setări pentru site specifice unei instanțe a siteului - sunt diferențe pe fiecare server.

- `submodules/`
  Acest folder conține submodule Git folosite de aplicația web.

  - `django-blog-zinnia/`
    Am făcut modificări la codul sursă al proiectului `django-blog-zinnia`. Am publicat modificările într-un depozit pe Github[19].
  - `imagestore/`
    Am făcut modificări la codul sursă al proiectului `imagestore`. Am publicat modificările într-un depozit pe Github[20].

[19] https://github.com/mihaisucan/django-blog-zinnia
[20] https://github.com/mihaisucan/imagestore

Capitolul 3

Funcționalități din aplicația web

În capitolele precedente am vorbit despre siteul Universității Agora, despre funcțiile lui în general, structura pe disc a codului sursă, cerințele tehnice și despre proiectele *open source* folosite. În ansamblu, aplicația web este flexibilă, ușor de modificat, se pot adăuga funcții noi în orice moment și se poate schimba interfața fără să fie necesare modificări în codul Python - există o separare clară și strictă între logică, interfață și design.

În acest capitol voi prezenta paginile importante din aplicația web, împreună cu detalii tehnice și informații despre secțiunile de administrare ale siteului.

3.1 Prima pagină

Prima pagină afișează o diaporamă ce are caracter promoțional, compusă din unele poze încărcate pe site. Aici sunt selectate doar pozele încărcate în galeria de poze, și doar cele ce au eticheta „diaporama". În acest fel administratorii siteului pot alege care poze apar pe prima pagină. Utilizatorul poate face click pe poză pentru a vedea varianta mai mare.

Pozele din diaporamă sunt decupate și redimensionate automat. Dacă se fac modificări de design la prima pagină, pozele se pot redimensiona din nou.

Sub diaporamă sunt listate ultimele știri și legături către studiile de licență ale Facultății de Drept și Științe Economice. Pe partea stângă se găsește o hartă a siteului minimală: meniurile și submeniurile principale.

Șabloanele HTML folosite în această pagină sunt:

- `pages/base.html`,
- `pages/welcome.html`,
- `welcome/first_news.html`,
- `welcome/other_news.html`.

Toate șabloanele se găsesc în folderul ***`site_project/templates`***.

Tot conținutul paginii este generat dinamic, fără a fi necesară editare manuală din partea administratorilor. Pagina nu are un *view* propriu - este doar o simplă pagină în CMS. Șablonul `pages/welcome.html` este unul cât se poate de specific: acesta folosește *template tags* care generează conținutul dinamic.

Pentru primele două știri șablonul `page/welcome.html` are următorul cod:

```
<ol id="welcome-first-news">
  {% get_recent_entries 2 "welcome/first_news.html" %}
</ol>
```

Funcția `get_recent_entries` este un *template tag* din `zinnia_tags`, care aparține aplicației `django-blog-zinnia`[1]. Șablonul `welcome/first_news.html` este foarte simplu:

```
{% load i18n %}
{% for entry in entries %}
  <li>
    <p class="post-title">
<a href="{{ entry.get_absolute_url }}">
{{ entry.title }}
</a>
    </p>
    <p class="post-excerpt">
{{ entry.html_content|striptags|truncatewords_html:60|safe }}
    </p>
    <p class="post-date"
        title="{{ entry.creation_date|date:"r" }}">
{{ entry.creation_date|date:"DATE_FORMAT" }}
    </p>
    <p class="post-read-more">
<a href="{{ entry.get_absolute_url }}">
```

[1] http://django-blog-zinnia.com

```
{% trans "read more" %}
</a>
    </p>
  </li>
{% endfor %}
```

Pentru noutățile afișate doar cu titlul lor există o chemare la funcția `get_recent_entries` care indică șablonul `welcome/other_news.html`. Acest șablon afișează doar `entry.title` cu legătură spre adresa articolului.

Meniurile și submeniurile afișate pe prima pagină sunt generate din șablonul `pages/welcome.html` astfel:

```
<ul id="welcome-menus">
  {% show_menu 0 1 1 1 %}
</ul>
```

Funcția `show_menu` este un *template tag* din `menu_tags` care aparține aplicației `django-cms`[2]. Parametrii dați indică numărul de submeniuri dorite pentru secțiunile active și inactive ale siteului.

Diaporama de pe prima pagină este generată tot din șablonul `pages/welcome.html`:

```
{% tagged_objects "diaporama" "imagestore.Image"
    "welcome_images" 10 "?" %}
<script type="text/javascript"><!--
var header_images = [
{% for image in welcome_images %}
{% thumbnail image.image "950x250" crop="center"
    as header %}
["{{ image.title }}",
"/{{ LANGUAGE_CODE }}{% url imagestore-image-in-album
album_id=image.album.pk pk=image.id %}",
 "{{ header.url }}",
],{% endthumbnail %}
{% endfor %}
];
// --></script>

{% compress js %}
<script type="text/javascript"
 src="{{ STATIC_URL }}js/welcome.js"></script>
{% endcompress %}
```

[2] http://django-cms.org

Funcția `tagged_objects` este un *template tag* din `tagging_tags`, ce aparține pachetului `django-tagging`[3] folosit de aplicația `imagestore`[4] cu care este făcută galeria de poze. Această funcție preia 10 obiecte cu eticheta `diaporama` care sunt din modelul `imagestore.Image`, sortate aleator. Lista de imagini este salvată în variabila `welcome_images`.

Funcția `thumbnail` este un *template tag* din pachetul `sorl-thumbnail`[5]. Această funcție redimensionează poza la rezoluția dorită.

Funcția `compress` este un *template tag* din pachetul `django-compressor`[6]. Aceasta este folosită pentru minificare/comprimarea codului JavaScript. În `js/welcome.js` este codul care face animația între poze. Librăria jQuery[7] este folosită pentru a facilita compatibilitatea între navigatoarele web.

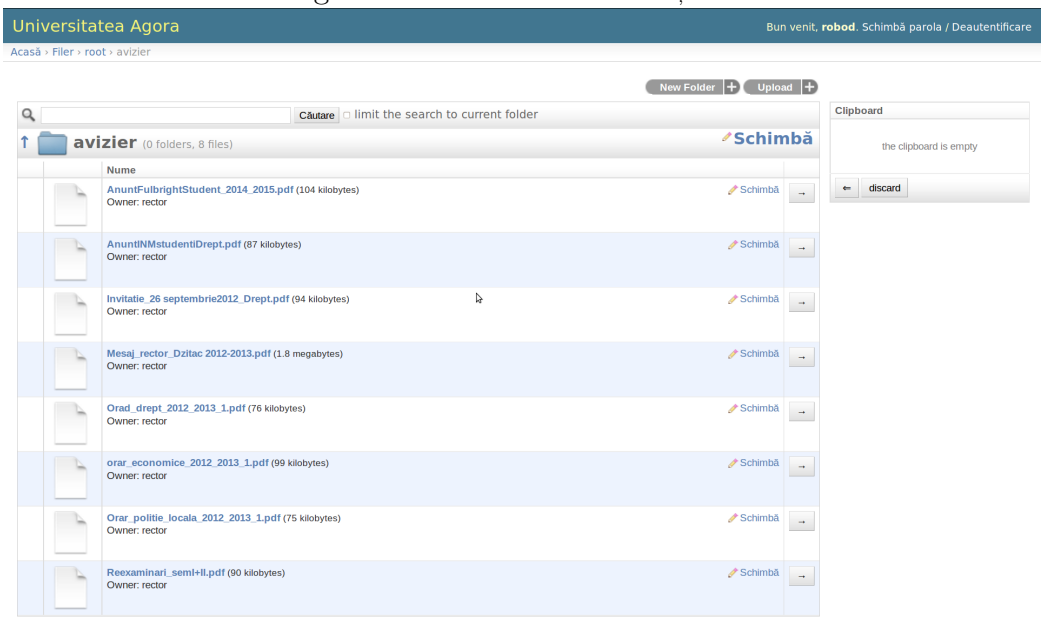
[3] https://code.google.com/p/django-tagging/
[4] https://github.com/hovel/imagestore
[5] https://github.com/sorl/sorl-thumbnail
[6] https://github.com/jezdez/django_compressor
[7] http://jquery.com

3.2 Gestionarea fișierelor de pe server

Gestionarul de fișiere, `django-filer`[8], permite utilizatorului să adauge foldere, fișiere, să facă redenumiri, să schimbe drepturile de acces, să facă operații de mutare/copiere/ștergere în masă, și se pot face căutări după fișierele dorite.

În cazul pozelor utilizatorii pot alege subiectul central, astfel încât să nu fie tăiat la decupări.

Figura 3.1: Gestionarul de fișiere.



După cum se poate vedea în captura de ecran există noțiunea de memorie de tampon, „*clipboard*”. Utilizatorul poate muta fișiere dintr-un folder în clipboard, adică o zonă temporară. Fiecare fișier are un buton cu săgeată spre dreapta - dacă se face click pe el, acesta este mutat în clipboard. După ce toate fișierele dorite sunt în clipboard, acestea pot fi mutate de acolo într-o altă destinație - un alt folder ales de utilizator. În acest mod operațiile de copiere/mutare în masă a fișierelor se pot face mult mai ușor.

Acest gestionar de fișiere este folosit și atunci când administratorul dorește să atașeze o poză sau un alt fișier într-o pagină. Administratorul trebuie doar să navigheze până la fișier și să facă click pe fișierul dorit pentru adăugare în articol.

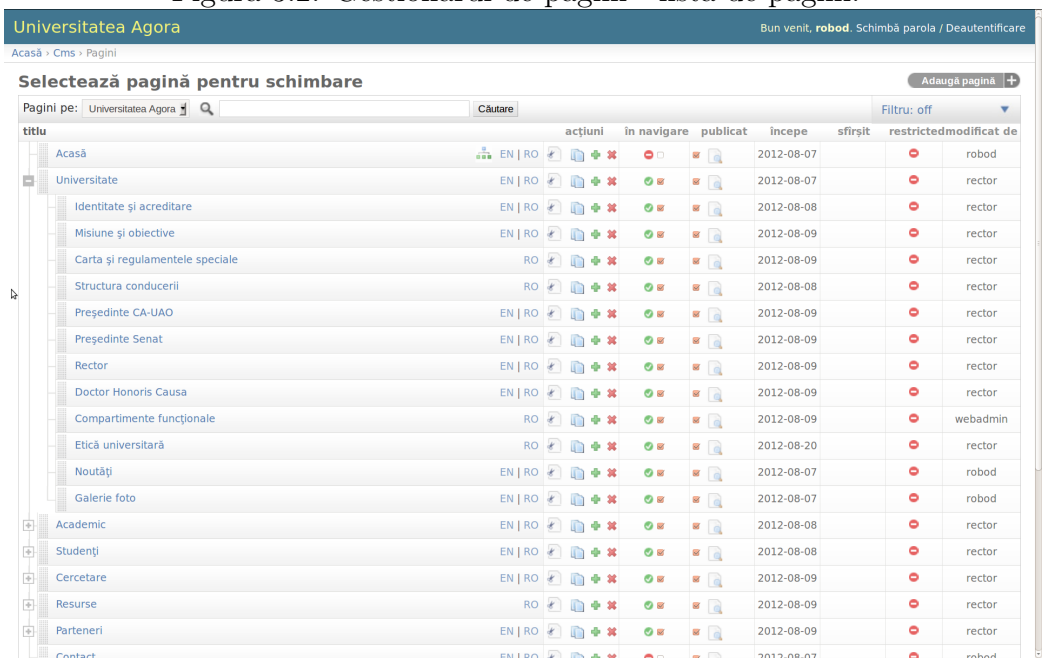
[8] https://github.com/stefanfoulis/django-filer

Django Filer aduce în proiect două câmpuri importante: `filer.fields.FilerImageField` și `FilerFileField`. Acestea sunt folosite în modelele Django care acceptă imagini și/sau alte fișiere.

3.3 Gestionarea paginilor

Gestionarul de pagini din `django-cms`[9] prezintă structura ierarhică a paginilor din site. După cum se poate vedea în captura de ecran, la fiecare pagină sunt indicatori pentru limbile în care aceasta este tradusă, data publicării, dacă pagina este publică, dacă apare în meniul de navigare și alte informații.

Figura 3.2: Gestionarul de pagini - lista de pagini.



Pentru a reordona paginile se poate face *drag* la titlu dintr-un loc în altul.

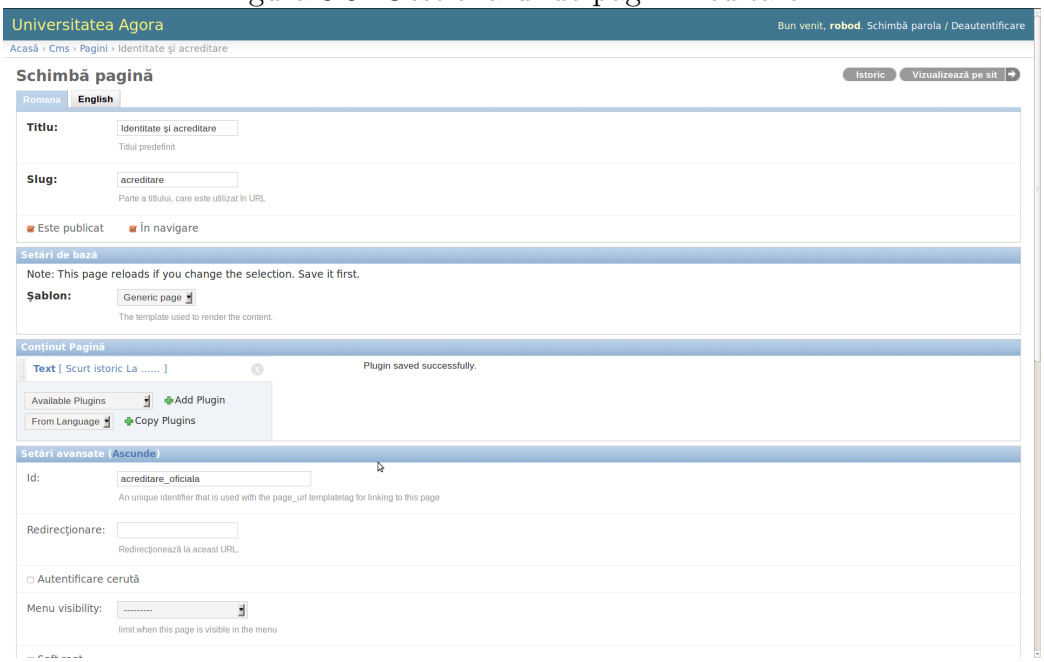
Dacă sunt multe pagini, utilizatorul poate să le filtreze în mai multe feluri: există un câmp de căutare, filtre după autor, limbă, starea paginii (publicată sau nu) și după alte criterii.

Fiecare pagină poate să aibă conținut, să fie doar o legătură spre o altă adresă sau poate să fie o aplicație. De exemplu, paginile „Noutăți” și „Galerie foto” sunt aplicații care au conținut propriu, generat dinamic din baza de date. Editarea conținutului la acestea se face din secțiuni diferite.

[9] http://django-cms.org

Pentru a edita o pagină utilizatorul trebuie doar să facă click pe titlul ei.

Figura 3.3: Gestionarul de pagini - editare.



În timpul editării unei pagini utilizatorul are opțiuni de adăugare conținut - poze, filme, fișiere, foldere, legături spre alte pagini, etc. De asemenea se pot face pagini care redirecționează vizitatorii spre alte pagini, se poate schimba interfața/șablonul paginii, se poate edita pagina în alte limbi, etc.

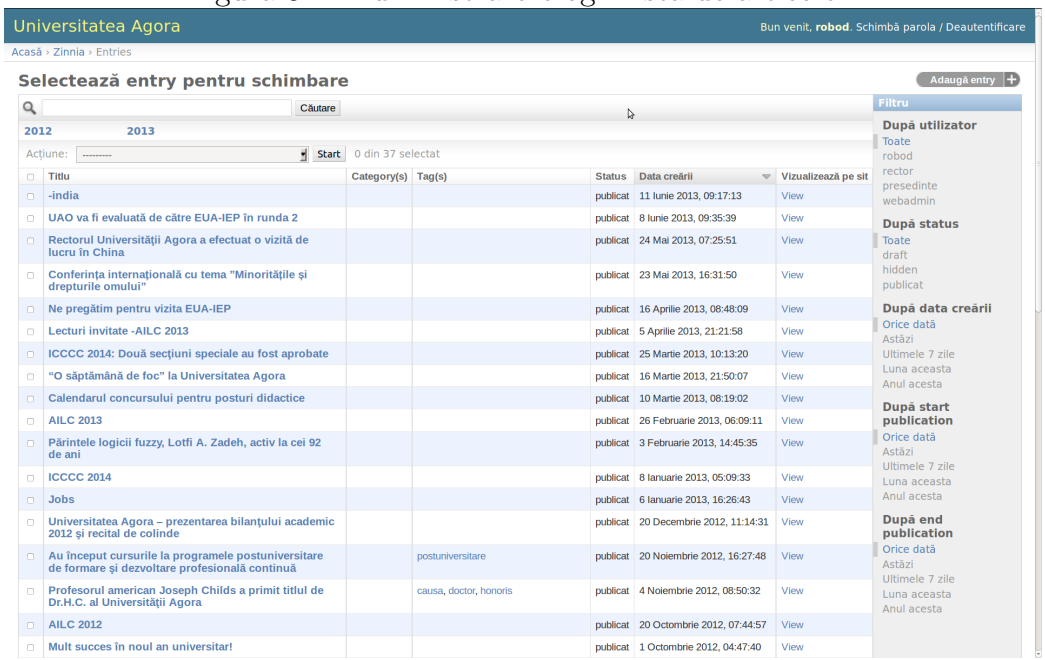
Fiecare șablon HTML de pagină în `django-cms` definește *placeholders* - regiuni ce pot fi editate de administratorii siteului. În aceste *placeholders* utilizatorul poate adăuga *plugins* - tipuri diferite de conținut, de exemplu text, fișier, poză, video, etc. Plugins sunt modele Python care moștenesc modelul `CMSPlugin`. O pagină în CMS este un model cu câmpurile pentru titluri în diferitele limbi, calea, setările, șablonul ales și lista de plugins adăugate în fiecare placeholder. Toate aceste informații se salvează în baza de date, inclusiv conținutul/setările introduse în interfața fiecărui plugin.

3.4 Blogul

Blogul „noutăți” din siteul Universității Agora permite vizitatorilor să filtreze conținutului după etichete și categorii. Aceștia se pot abona la știri folosindu-se de fluxul Atom pus la dispoziție - există programe specializate care oferă

asemenea opțiuni, de exemplu Mozilla Thunderbird[10]. Pentru a face căutarea articolelor mai ușoară am inclus și pagini de arhivă unde vizitatorii pot vedea articolele grupate după ani sau luni.

Figura 3.4: Administrare blog - lista de articole.



Pentru administratori lista de articole poate fi filtrată după titlu, an, autor, status și alte criterii.

Editarea bilingvă este cât mai ușoară: utilizatorul poate vedea ambele variante ale articolului în același timp. Articolele pot fi adăugate sub forma de „ciorne" - aceste articole nu apar pe site până când autorul alege să le facă publice. Articolele pot avea și statutul „ascuns": un articol ascuns este public, poate fi accesat, dar nu apare în lista de articole pe blog.

Fiecare articol pe blog poate avea o poză asociată - aceasta este redimensionată automat. Vizitatorii pot face click pe poză pentru a vedea varianta originală. De asemenea, la fiecare articol autorul poate adăuga o listă de articole similare / *related entries*. Pentru protecție articolele pot fi parolate.

Implementarea blogului este făcută cu pachetul `django-blog-zinna`[11]. Principalele modele definite de acest proiect sunt: `zinnia.models.Author`, `Category` și `Entry`. Fiecare articol este asociat cu zero sau mai multe categorii, și poate avea unul sau mai mulți autori.

[10] https://www.mozilla.org/thunderbird/
[11] http://django-blog-zinnia.com

Șablonul de bază folosit pentru paginile din blog este `site_project/templates/zinnia/base.html`. Listarea de articole se face cu `entry_list.html` din același folder. Un articol de sine stătător este afișat cu ajutorul șablonului `entry_detail.html`.

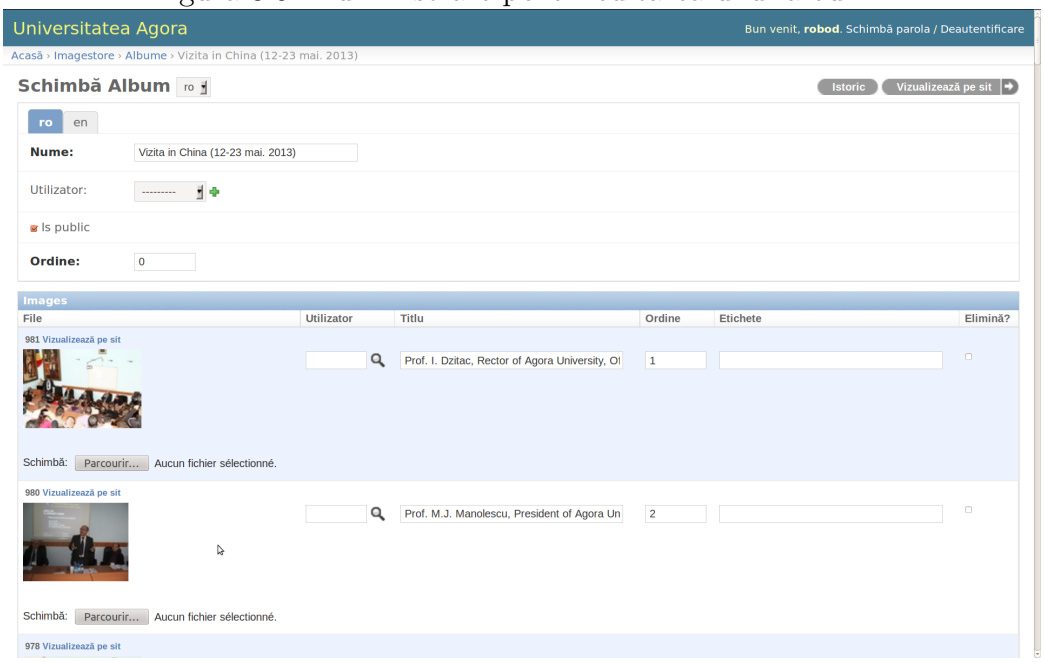
Codul pentru integrarea aplicației de blog cu `django-cms` se găsește în `apps/robo/blog/cms_app.py`.

Fiecare *view* este generat cu ajutorul claselor generice pentru *views* - de exemplu, `django.views.generic.list_detail.object_list` pentru prima pagină a blogului, sau `date_based` *views* pentru arhivele de articole.

3.5 Galeria de imagini

Galeria de poze este structurată pe albume și etichete prin care utilizatorii pot naviga în orice fel își doresc. Fiecare poză are titlu, etichete și o descriere. Vizitatorii siteului pot vedea și varianta originală - fără redimensionare.

Figura 3.5: Administrare poze - editarea unui album.



Administratorii pot încărca poze în masă, un fișier ZIP cu toate pozele. La fiecare poză se pot adăuga descriere, titlu și etichete. Fiecare album poate să fie public sau nu - până când autorul îl finalizează și alege să-l facă public.

După cum se poate vedea în captura de ecran de mai sus, editarea unui album este optimizată pentru multe poze. Se pot face modificări la mai multe poze simultan.

Pentru implementarea galeriei de poze am folosit pachetul *open source* `imagestore`[12]. Imaginile sunt reprezentate de modelul `imagestore.models.Image`. Acesta folosește câmpul `ImageField` din `sorl-thumbnail`[13] pentru a gestiona mai ușor fișierele de tip imagine. Albumele au modelul lor: `imagestore.models.Album`.

Pentru *views* `imagestore` folosește views generice din Django: `ListView` și `DetailView`.

Șabloanele HTML folosite pentru galeria de poze:

- `imagestore/image.html` pentru vizualizarea unei poze.
- `imagestore/image_list.html` pentru lista de poze.
- `imagestore/album_list.html` pentru lista de albume.

[12] https://github.com/hovel/imagestore
[13] https://github.com/sorl/sorl-thumbnail

3.6 Pagina Contact

Pagina de contact prezintă informații generale de contact și un formular de contact unde vizitatorii pot să scrie un mesaj către departamentul dorit. Formularul este protejat de spam cu ajutorul serviciului recaptcha.net.

Figura 3.6: Pagina de contact.

Administratorii pot adăuga adrese/departamente la care utilizatorii pot trimite mesaje. De asemenea ei pot edita informațiile de contact afișate în pagină.

Implementarea acestui formular se găsește în folderul `apps/robo/contact`. Am definit două modele `ContactRecipient` și `ContactGroup`:

```
from django.db import models

class ContactGroup(models.Model):
    title = models.CharField(_('title'), \
                             max_length=150)

    class Meta:
        verbose_name = _('contacts group')
        verbose_name_plural = _('groups of contacts')
```

```
    def __unicode__(self):
        return self.title

class ContactRecipient(models.Model):
    group = models.ForeignKey(ContactGroup, \
                              verbose_name=_('group'))
    display_name = models.CharField(_('name'), \
                                    max_length=150)
    email = models.EmailField(_('email'), \
                              max_length=100)

    class Meta:
        verbose_name = _('recipient')
        verbose_name_plural = _('recipients')
        unique_together = (('group', 'email'), \
                           ('group', 'display_name'))

    def __unicode__(self):
        return self.display_name
```

Pentru integrarea cu `django-cms`[14] am făcut un model `CMSPlugin` care poate fi folosit ca plugin în orice pagină, în orice *placeholder* din oricare șablon.

```
from cms.models import CMSPlugin

class ContactPluginModel(CMSPlugin):
    group = models.ForeignKey(ContactGroup, null=True,
            on_delete=models.SET_NULL, \
            verbose_name=_('group'))

    def copy_relations(self, oldinstance):
        self.group = oldinstance.group

    def __unicode__(self):
        if self.group:
            return self.group.title
        return _('<no recipients>')
```

În `cms_plugin.py` se găsește și implementarea modelului `ContactPlugin` care afișează formularul de contact și procesează datele introduse de utiliza-

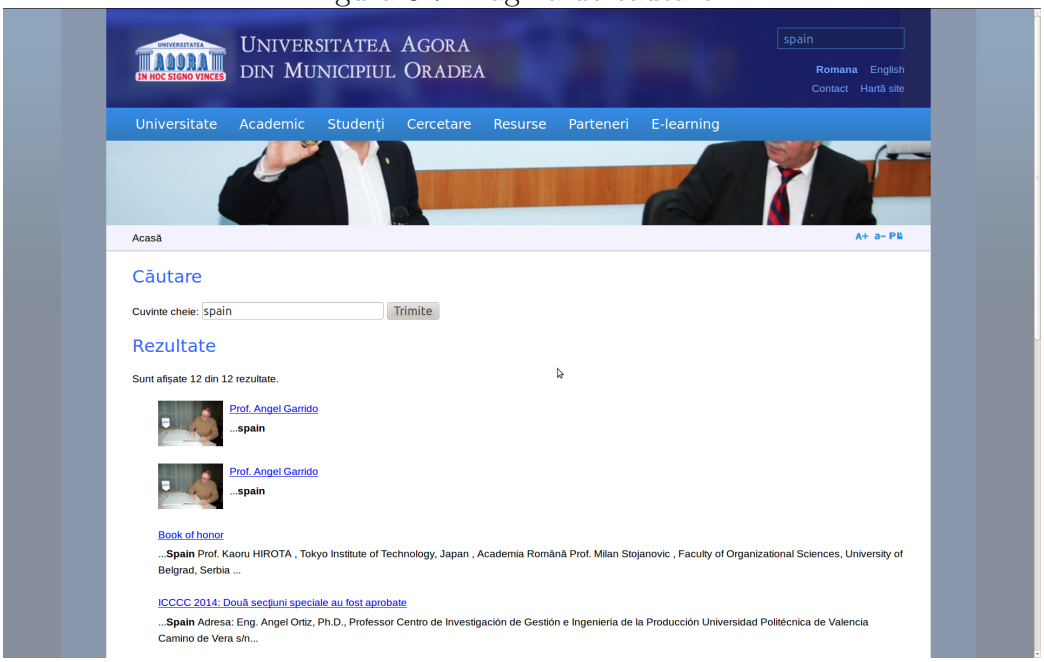
[14] http://django-cms.org

tor. Șablonul HTML folosit de formular: `templates/contact/contact.html`.

3.7 Motorul de căutare

Motorul de căutare al siteului caută în conținutul paginilor, articolelor din blog și în descrierile pozelor. Pozele sunt listate nu doar ca legături, se face și o previzualizare a pozei.

Figura 3.7: Pagina de căutare.



Pentru implementarea motorului de căutare sunt folosite următoarele pachete *open source*:

Whoosh este un motor de indexare și de căutare documente, scris în Python. [15]

django-haystack este un pachet care integrează Whoosh și alte motoare de căutare cu Django. Acesta abstractizează modelele de indexare a fiecărui motor de căutare, și oferă funcții de căutare comune pentru ele. [16]

[15] https://bitbucket.org/mchaput/whoosh/
[16] http://haystacksearch.org

django-cms-search [17] este un pachet care include modele de indexare a conținutului din paginile `django-cms`[18]. Acestea sunt modele `haystack.indexes.SearchIndex`, specifice `django-haystack`.

Fiecare secțiune din siteul Universității Agora are un model, un tabel în baza de date: există modelele pentru pagini, articolele din blog și pentru pozele din galerie - așa cum au fost explicate în secțiunile anterioare ale acestui document. Pentru fiecare model există încă un model `SearchIndex` care descrie cum trebuie indexat modelul din baza de date. Acestea se găsesc în fișierele `search_indexes.py` din folderele pachetelor Django. Mai jos redau modelul `SearchIndex` pentru blog din fișierul `apps/robo/blog/search_indexes.py`:

```
# ...
from haystack import indexes
from zinnia.models import Entry
from robo.utils.search \
    import register_multilanguage_search

class EntryIndex(indexes.SearchIndex):
    text = indexes.CharField(document=True, \
                             use_template=False)
    pub_date = indexes.DateTimeField( \
                model_attr='start_publication', \
                null=True)
    login_required = indexes.BooleanField( \
                        model_attr='login_required')
    url = indexes.CharField( \
            stored=True, \
            indexed=False, \
            model_attr='get_absolute_url')
    title = indexes.CharField(stored=True, \
                              indexed=False, \
                              model_attr='title')
    site_id = indexes.IntegerField(stored=True, \
                                   indexed=True)

    def prepare(self, obj):
        self.prepared_data = super(EntryIndex, self) \
```

[17] https://github.com/piquadrat/django-cms-search
[18] http://django-cms.org

```
                                .prepare(obj)

        title = force_unicode(obj.title)
        content = _strip_tags(obj.html_content)
        self.prepared_data['text'] = title + "\n" + \
                force_unicode(obj.tags) + \
                "\n" + content

        site_id = Site.objects.get_current().id;
        if obj.sites.count() > 0:
            site_id = obj.sites.get().id
        self.prepared_data['site_id'] = site_id

        return self.prepared_data

    def index_queryset(self):
        return self.model.published.all()

register_multilanguage_search(Entry, EntryIndex)
```

După cum se poate vedea mai sus sunt definite câmpurile de indexare pentru motorul de căutare. Acestea sunt tot timpul aceleași, în toate secțiunile din site. Ceea ce diferă de fiecare dată este felul în care modelul Django este translatat în `SearchIndex`. În acest exemplu se vede cum se face indexarea pentru `zinnia.models.Entry` - acestea reprezintă articolele din blog.

Pentru suportul multilingvistic al aplicației am folosit `django-modeltranslation`[19]. Cu ajutorul acestuia a introdus traduceri la câmpurile fiecărui model din aplicație. Modelele de traducere se găsesc în fiecare fișier numit `translation.py`.

Fiecare pagină, în fiecare limbă, este indexată separat pentru motorul de căutare. Cu funcția `register_multilanguage_search()` din `robo.utils.search` fac mai simplă înregistrarea fiecărui model, în fiecare limbă disponibilă.

Pentru pagina de căutare este folosit șablonul `site_project/templates/search/search.html`.

[19] https://github.com/deschler/django-modeltranslation

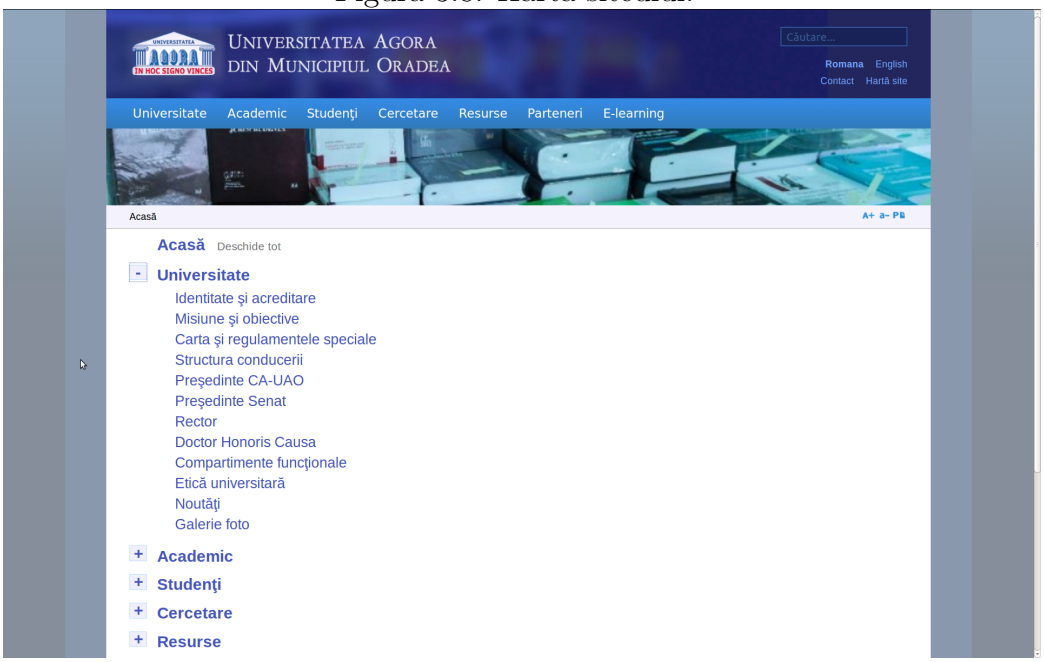
3.8 Harta siteului

Harta siteului este generată dinamic cu ajutorul pachetului inclus implicit în Django: `django.contrib.sitemaps`[20]. Harta este disponibilă în format Sitemaps XML [1]. Pentru fiecare secțiune din site există un fișier `sitemaps.py` care include o clasă `Sitemap` ce definește felul în care fiecare model Django este indexat. Mai jos redau clasa `Sitemap` folosită pentru indexarea pozelor și a albumelor din site[21]:

```
from django.contrib.sitemaps import Sitemap
from django.core.urlresolvers import reverse
from imagestore.models import Album, Image

class ImageSitemap(Sitemap):
    priority = 0.5
    changefreq = 'monthly'

    def items(self):
        return Image.objects.filter( \
                album__is_public=True)
    def lastmod(self, obj):
        return obj.updated
    def location(self, obj):
        return reverse('imagestore-image-in-album', \
                kwargs={'album_id': obj.album.pk, \
                        'pk': obj.pk})

class AlbumSitemap(Sitemap):
    priority = 0.5
    changefreq = 'monthly'
    def items(self):
        return Album.objects.filter( \
                is_public=True, \
                head__isnull=False)
    def lastmod(self, obj):
        return obj.updated
```

[20] https://docs.djangoproject.com/en/1.3/ref/contrib/sitemaps/
[21] vezi fișierul `apps/robo/gallery/sitemaps.py`

Figura 3.8: Harta siteului.



Pentru pagina HTML pachetul *open source* `cmsplugin-htmlsitemap`[22] este folosit. Acesta este un *plugin* pentru `django-cms` ce generează o hartă a paginilor, în fiecare limbă.

Pagina cu harta siteului folosește șablonul `site_project/templates/pages/sitemap.html`. Pentru interactivitate este folosit fișierul JavaScript: `site_project/static/js/sitemap.js`.

[22] https://github.com/raphaa/cmsplugin-htmlsitemap

Concluzii

Siteul Universității Agora este acum un site modern, cu design atractiv și cu o structură ușor de navigat. Această aplicație web modernă este dezvoltată într-un mod ce va permite proprietarilor să crească siteul de la an la an, adăugând conținut cât și funcții noi. Alegerea platformei Django a fost o decizie esențială care va avea un impact mare în timp - colaboratori noi, programatori noi, echipe interesate din cadrul universității au posibilitatea să itereze îmbunătățiri de conținut și modificări tehnice la site.

Proiectul prezentat în această lucrare trebuie să evolueze cu ajutorul studenților pasionați de informatică. Cu ajutorul tehnologiei Git se pot face modificări la site, în colaborare. Pe viitor se pot face optimizări tehnice, încărcare mai rapidă a conținutului și alte modificări.

Mulțumesc domnului profesor coordonator Ioan Dzițac pentru oportunitatea de a realiza o aplicație web modernă, și pentru sprijinul acordat în completarea acestuia cu conținut. Mulțumesc și lui Marius Șucan pentru realizarea interfeței pentru site.

Glosar

Apache este cel mai folosit server web[1], distribuit ca software liber de Fundația Apache. 3, 9

Atom este un format XML standard, RFC 4287[2], pentru fluxuri web. Acesta permite navigatoarelor web sau oricărui alt software să verifice dacă sunt actualizări la resurse de pe web. În unele implementări mai avansate, formatul Atom poate permite și publicarea și actualizarea resurselor/paginilor web. 6, 9, 24

CMS (Content Management System) este un sistem de gestiune a conținutului. 19, 24

CSS (Cascading Style Sheets) este un limbaj de formatare a elementelor dintr-o pagină web[3], standardizat de către W3C. 7, 9

Django este un cadru pentru dezvoltarea aplicațiilor web cu Python [4]. Acest proiect are sursa liberă și este distribuit sub licență BSD[5]. 8, 10, 11, 35

ECMAScript este un limbaj de programare dinamic standardizat de Ecma International, în cadrul specificației ECMA-262[6] și ISO/IEC 16262. Limbajul este implementat sub forma unor dialecte bine cunoscute precum JavaScript, JScript și ActionScript. 37

Git este un sistem distribuit de gestionare a codului sursă[7]. Acest proiect are sursa liberă și este distribuit sub licență GPL[8] v2. 3, 8, 9, 17, 35

[1] http://httpd.apache.org
[2] http://tools.ietf.org/html/rfc4287
[3] http://www.w3.org/Style/CSS/
[4] http://djangoproject.com
[5] Berkley Software Distribution
[6] http://www.ecma-international.org/publications/standards/Ecma-262.htm
[7] http://git-scm.org
[8] GNU General Public License

HTML (HyperText Markup Language) este un limbaj de marcare folosit pentru crearea paginilor web[9], standardizat de către W3C. Navigatoarele web interpretează codul HTML pentru a afișa conținutul. 7–9, 11, 17, 18, 24, 27, 30, 34, 37

HTTP (HyperText Transfer Protocol) este un protocol de rețea distribuit pentru colaborare în sisteme informatice interactive [3]. Acest protocol este de tip cerere – răspuns, unde clientul, navigatorul web, cere o anumită adresă și primește răspunsul de la server. HTTP este fundamentul webului, fiind principalul protocol de transfer de date. 10, 13

JavaScript este o implementare a standardului ECMAScript, ce se regăsește cu precădere în navigatoarele web. Acest limbaj de programare permite aplicațiilor web să fie dinamice și interactive. 7, 9, 34

Linux este un sistem de operare gratuit cu sursă liberă a cărui componentă principală este nucleul monolitic care are același nume: Linux [5].. 3, 9

MySQL este un sistem de gestiune a bazelor de date relațional[10], software liber distribuit sub licența GPL[11] v2. 3, 8–10, 12

nginx este un server web mic și performant[12], distribuit sub licență BSD[13]. 3, 9

PHP este un limbaj de programare dinamic[14] ce nu are o specificație formală. Acesta este folosit cu precădere pe serverele web pentru generarea dinamică a paginilor HTML. Principala implementare a limbajului este un software liber cunoscut tot sub numele de PHP. 10

PostgreSQL este un sistem de gestiune a bazelor de date relațional[15], software liber.. 3, 8, 9, 12

Python este un limbaj de programare dinamic [16]. 3, 6, 8, 9, 13, 24, 30

[9] http://www.w3.org/html/wg/
[10] http://www.mysql.com
[11] GNU General Public License
[12] http://nginx.org
[13] Berkley Software Distribution
[14] http://www.php.net
[15] http://www.postgresql.org
[16] http://www.python.org

SQLite este un sistem de gestiune a bazelor de date [17]. Codul sursă al acestui proiect este în domeniul public - fără nici o restricție de licențiere.. 3, 8, 12

URL (*Uniform Resource Locator*) înseamnă localizator uniform de resurse. Un URL este o referință pentru o resursă. Aceste URL-uri sunt cunoscute și sub numele de adrese web. [4]. 10, 13

W3C (World Wide Web Consortium) este principalul consorțiu internațional de standardizare a webului[18]. Membrii organizației dezvoltă și ratifică standarde web precum HTML, CSS și SVG. 36–38

Whoosh este o librărie de indexare și de căutare implementată în Python [19]. Acest proiect are sursa liberă și este distribuit sub licență BSD[20]. 6, 9

WSGI (Web Server Gateway Interface) este o interfață simplă și universală de comunicare între un server web și o aplicație web făcută în Python. [2]. 3, 9, 10, 14, 16

XML (Extensible Markup Language) este un meta-limbaj de marcare, de descriere structurată a datelor[21], standardizat de către W3C. Cu ajutorul acestui meta-limbaj se pot defini alte limbaje de marcare, precum XHTML, Atom și SVG. 9

[17] http://www.sqlite.org
[18] http://www.w3.org
[19] https://bitbucket.org/mchaput/whoosh/
[20] Berkley Software Distribution
[21] http://www.w3.org/XML/

Listă de figuri

Bibliografie

[1] Documentația Sitemaps XML.
http://www.sitemaps.org/

[2] Web Server Gateway Interface.
https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface

[3] Descrierea protocolului HTTP.
http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[4] Uniform Resource Locator.
https://en.wikipedia.org/wiki/Uniform_resource_locator

[5] Sistemul de operare Linux.
https://en.wikipedia.org/wiki/Linux

[6] Documentația Django.
https://docs.djangoproject.com/

[7] Documentația Python.
http://docs.python.org/2.7/