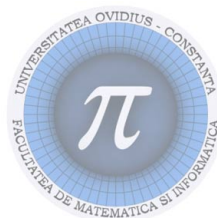


UNIVERSITATEA OVIDIUS



FACULTATEA DE MATEMATICĂ ȘI  
INFORMATICĂ

LUCRARE DE LICENȚĂ

---

# Dezvoltare aplicație web SPA bazată pe Python/Django și AngularJS

---

*Student:*  
Ștefan Daniel MIHĂILĂ

*Profesor îndrumător:*  
Lect. dr. Andrei RUSU

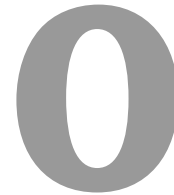
iulie 2015

# Cuprins

<b>0</b>	<b>Introducere</b>	<b>2</b>
<b>1</b>	<b>Aplicații SPA: moduri de dezvoltare</b>	<b>4</b>
1.1	Framework-uri MVC JavaScript . . . . .	4
1.2	AJAX . . . . .	5
1.3	WebSocket . . . . .	5
1.4	Plugin-uri pentru browser . . . . .	6
<b>2</b>	<b>Python și Django</b>	<b>7</b>
2.1	Python . . . . .	7
2.2	Django . . . . .	10

*Internetul este un amalgam de tehnologii, legate împreună cu bandă adezivă, sfoară și gumă de mestecat. Nu este ceva proiectat într-un mod elegant, pentru că este un organism în creștere, nu o mașinărie construită cu intenție.*

Mattias Petter Johansson (Programator la Spotify)



# Introducere

Internetul a evoluat continuu și a ajuns în punctul în care poate face o mulțime de lucruri pentru care nici măcar nu a fost creat. Aproape toți programatorii din ziua de azi sunt programatori web, iar aplicațiile web seamănă tot mai mult cu aplicațiile desktop. În aceste condiții, a devenit foarte important pentru dezvoltatori să poată crea astfel de aplicații într-un mod rapid și eficient, iar uneltele pe care le au la dispoziție au fost reinnoite permanent cu altele mai bune.

Arhitectura web clasică este una client-server, în care clientul (browserul) cere o pagină folosind protocolul HTTP, serverul o crează dinamic folosind un limbaj de programare server-side (C#, Java, Python, PHP, Scala etc.) și o trimite browserului pentru afișare. Prin HTTP, conexiunile sunt întotdeauna inițiate de către client, care cere pagina web.

Această arhitectură este limitată. Să ne imaginăm de exemplu că avem o pagină web care afișează în timp real scorurile unor partide de fotbal. După încărcarea paginii, server-ul nu-i poate comunica browserului că un scor s-a schimbat. Browserul va afișa scorurile neactualizate până când utilizatorul reîmprospătează pagina.

Această problemă a fost rezolvată prin intermediul AJAX<sup>1</sup>, o tehnică ce permite browserului să facă cereri asincrone către server după ce pagina a fost încărcată, prin intermediul JavaScript.

Următoarea etapă în acest proces incremental a fost crearea de *Single-Page Application*<sup>2</sup>, denumite în continuare SPA. Într-un SPA, tot codul HTML, JavaScript și CSS este fie descărcat în momentul în care pagina este încărcată prima dată, fie în mod asincron, de obicei ca răspuns la acțiunile utilizatorului.

SPA oferă utilizatorului senzația unei aplicații fluide și poate uneori să ofere iluzia că aceasta răspunde la acțiuni imediat, fără să mai aștepte răspunsul serverului. Vom vedea în aplicația construită pentru această lucrare, de exemplu, că atunci când utilizatorul dorește ștergerea unei resurse, această resursă este

---

<sup>1</sup>Asynchronous JavaScript and XML; în aplicațiile moderne se utilizează cu preferință JSON (JavaScript Object Notation) în loc de XML, dar denumirea a rămas.

<sup>2</sup>[http://en.wikipedia.org/wiki/Single-page\\_application](http://en.wikipedia.org/wiki/Single-page_application)

întâi înlăturată din UI, apoi o cerere asincronă îi spune serverului să șteargă resursa din baza de date. Desigur, pentru că se comunică cu serverul prin TCP/IP, această comunicare poate eșua, caz în care un mesaj de eroare este afișat și resursa re apare în UI, dar în mai mult de 90% din cazuri, când totul merge bine, utilizatorul are senzația că resursa este ștearsă instant.

Două companii foarte mari, Google și Facebook, au creat fiecare câte un framework pentru crearea de SPA: AngularJS și React, confirmând importanța acestui tip de aplicații. Experiență fluidă pentru utilizator, împreună cu alte avantaje pe care le vom discuta în capitolul următor au făcut ca SPA să crească foarte mult în popularitate în ultimii ani.

*Cândva oamenii credeau că internetul este o altă lume,  
dar acum realizează că este o unealtă pe care o folosim în  
lumea noastră.*

Tim Berners-Lee, inventatorul *www*-ului

# 1

## Aplicații SPA: moduri de dezvoltare

În acest capitol vom enumera diferite moduri în care se pot dezvolta SPA și vom discuta avantajele și dezavantajele acestor moduri.

### 1.1 Framework-uri MVC JavaScript

Anumite framework-uri JavaScript pentru creare de aplicații web, cum ar fi Backbone.js<sup>1</sup>, AngularJS<sup>2</sup>, Ember.js<sup>3</sup>, React<sup>4</sup> și Meteor<sup>5</sup> și-au propus să ușureze dezvoltarea de aplicații web SPA.

Aceste framework-uri oferă de obicei și posibilitatea organizării codului folosind șablonul arhitectural *Model-view-controller*<sup>6</sup> (MVC). MVC a fost folosit inițial în dezvoltarea aplicațiilor desktop, dar s-a dovedit mai târziu util și pentru dezvoltarea părții de back-end (server-side) a aplicațiilor web. Abia de curând el a fost adoptat și pe front-end (client-side).

MVC decuplează datele și logica aplicației de prezentare (interfața cu utilizatorul). Într-un framework JavaScript MVC, view-ul este reprezentat de șabloane HTML, controller-ul este un obiect JS care se ocupă de comunicarea dintre view și model, iar modelul este un obiect JS care, de obicei, mapează obiectele din baza de date de pe server la obiecte afișate de interfața cu utilizatorul. Desigur, o aplicație ce rulează în browser nu are acces în mod direct la baza de date, de aceea această mapare se face apelând un API REST<sup>7</sup>.

În continuare enumerăm câteva avantaje ale folosirii unui framework JS pentru dezvoltarea de aplicații SPA:

---

<sup>1</sup><http://backbonejs.org/>

<sup>2</sup><https://angularjs.org/>

<sup>3</sup><http://emberjs.com/>

<sup>4</sup><http://facebook.github.io/react/>

<sup>5</sup><https://www.meteor.com/>

<sup>6</sup><http://en.wikipedia.org/wiki/Model-view-controller>

<sup>7</sup>[http://en.wikipedia.org/wiki/Representational\\_state\\_transfer](http://en.wikipedia.org/wiki/Representational_state_transfer)

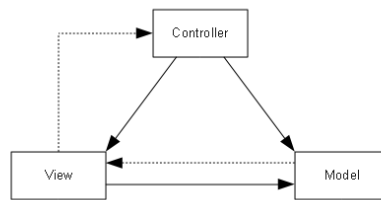


Figura 1.1: Interacțiunea dintre componentele MVC

- Folosirea MVC: un proiect MVC este mai ușor de navigat, este mai ușor de modificat și mai ușor de înțeles. De asemenea, colaborarea dintre designer și programator este ușurată de MVC.
- Viteza de dezvoltare (după depășirea curbei de învățare).
- Ușurința de dezvoltare.
- Diminuarea codului necesar a fi scris.

Bineînțeles, există și dezavantaje:

- Necesitatea învățării unei tehnologii noi. Ce este și mai trist, este că foarte posibil această tehnologie va fi depășită în doar câțiva ani.
- Unele framework-uri (AngularJS) au o curbă de învățare abruptă.
- Frameworkurile au tendința de a nu suporta browserele mai vechi.

## 1.2 AJAX

AJAX este modalitatea "clasică" prin care pot fi create SPA. Avantaje:

- Fiind o modalitate mai veche, este cunoscută de mai mulți programatori.
- Suport mai bun pentru browserele vechi.

Dezavantaje:

- Este necesar să se scrie mult cod.
- Folosirea MVC este mai dificilă.
- Viteză mică de dezvoltare.

## 1.3 WebSocket

WebSocket<sup>8</sup> este un protocol ce permite comunicare bidirecțională cu serverul. Atunci când clientul inițiază comunicarea cu serverul prin HTTP, serverul îi poate cere clientului să treacă la WebSocket. Dacă trecerea are loc, atunci

<sup>8</sup><http://en.wikipedia.org/wiki/WebSocket>

serverul îi poate trimite notificări clientului, lucru care nu este posibil pe HTTP. Performanța WebSocket este mai bună decât AJAX. În plus, soluția este mai elegantă. Cu AJAX, clientul face *polling*, adică întreabă serverul la un anumit interval de timp dacă informații noi sunt disponibile. Cu WebSocket, nevoia pentru polling este eliminată.

WebSocket este un protocol relativ nou și este implementat doar pe browserele moderne. Pentru browserele mai vechi există librării JS care simulează WebSocket folosind AJAX.

WebSocket și frameworkurile JS pentru SPA nu sunt mutual exclusive. De exemplu, Meteor folosește WebSocket atunci când clientul suportă acest protocol și *SockJS*<sup>9</sup> atunci când protocolul nu este suportat.

## 1.4 Plugin-uri pentru browser

*Java Applet*-urile, o tehnologie creată de *Sun Microsystems* (cumpărat de Oracle) au promis prin anii '90 că vor revoluționa modul de dezvoltare al aplicațiilor web, dar nu au reușit să se țină de promisiune. Rata lor de adopție a rămas foarte mică.

*Macromedia*, ulterior achiziționat de *Adobe* a reușit să facă o treabă mult mai bună cu *Flash*, care a atins o rată de adopție mult mai mare, dar această rată este în prezent în continuă scădere. Unul din motivele începutului sfârșitului pentru Flash a fost decizia companiilor *Google* și *Apple* de a nu suporta plugin-ul pe platformele lor mobile (*Android* și *iOS*). În prezent cel mai cunoscut site care folosește Flash este YouTube, dar acesta oferă și o alternativă bazată pe HTML5.

*Microsoft* a creat un rival pentru Flash, și anume *Silverlight*, dar tehnologia a rămas aproape necunoscută, singurul site mare care a adoptat această tehnologie fiind *NetFlix*.

Aceste plugin-uri ușurau crearea de aplicații web complexe pentru dezvoltatori, dar desigur că era și un preț de plătit:

- Cerințe mari de resurse pentru calculatorul utilizatorului.
- Controlate de o companie.
- Closed-source.
- Interoperabilitate scăzută (suport scăzut pentru Linux, platforme mobile, browsere mai puțin cunoscute).

Din fericire, industria web "a decis" într-un mod organic să sprijine *Open Web*<sup>10</sup>, iar în prezent chiar și companiile din spatele acestor plugin-uri suportă, în ceea ce privește web-ul cel puțin, tehnologiile open-source și interoperabilitatea.

---

<sup>9</sup><http://sockjs.org>

<sup>10</sup>[http://en.wikipedia.org/wiki/Open\\_Web](http://en.wikipedia.org/wiki/Open_Web)

În comunitatea Python, a spune despre ceva că este isteț,  
nu este considerat un compliment.

Alex Martelli

# 2

## Python și Django

În scurta mea carieră de inginer software, am lucrat deja cu trei limbaje de programare: *C#*, *Scala* și *Python*. În acest capitol voi explica de ce am ales Python pentru dezvoltarea back-end-ului acestui proiect.

### 2.1 Python

Python este un limbaj dinamic, interpretat care pune accent pe lizibilitate.

Avantaje:

- Foarte ușor de învățat și folosit, poți deveni productiv în doar câteva zile.
- Comunitate puternică, deci se găsesc cu ușurință o mulțime de librării și framework-uri bine scrise și bine documentate și soluții la problemele comune.
- Comunitatea și filosofia<sup>1</sup> limbajului au pus mare accent pe lizibilitatea codului. Codul Python este foarte ușor de înțeles, chiar și de cineva fără experiență cu acest limbaj, iar limbajul permite scrierea codului într-un mod elegant, concis și expresiv.

Dezavantaje:

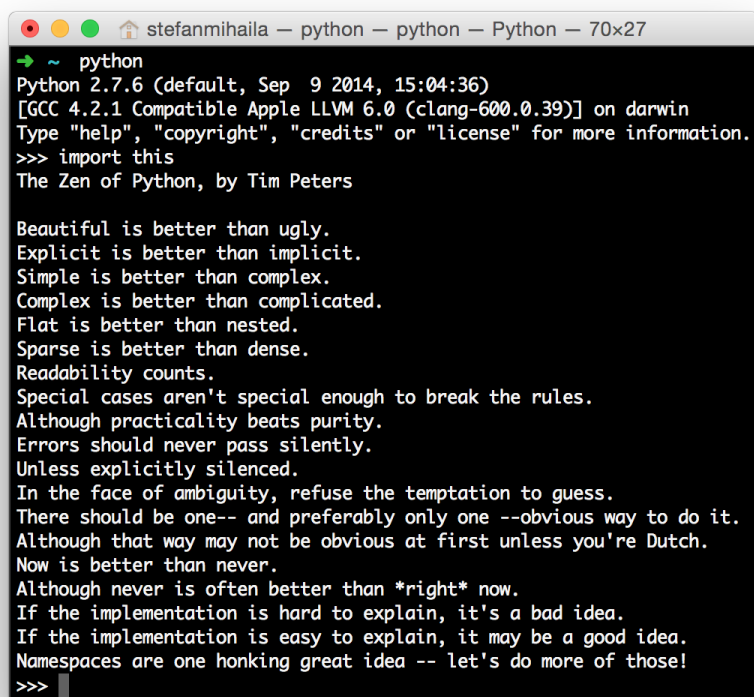
- Fiind un limbaj dinamic, IDE-ul nu înțelege la fel de bine structura codului, iar refactorizările se fac mai greu decât în limbajele statice.
- Este mai încet decât limbajele compilate.

În continuare, pentru comparare, prezint codul sursă al unui *web crawler*<sup>2</sup> foarte simplu, scris mai întâi în Java, apoi în Python. Codul este inspirat din *Java vs Python Platforms Comparison*, ce poate fi văzut la <https://www.youtube.com/watch?v=ppspz2ZiBaY>.

<sup>1</sup>Vezi Figura 2.1

<sup>2</sup>[http://en.wikipedia.org/wiki/Web\\_crawler](http://en.wikipedia.org/wiki/Web_crawler)



A screenshot of a macOS terminal window. The title bar at the top shows three colored window control buttons (red, yellow, green) followed by the text 'stefanmihaila — python — python — Python — 70x27'. The terminal content shows a shell prompt '~' followed by the command 'python'. This launches the Python 2.7.6 interpreter, which displays its version, build date (Sep 9 2014, 15:04:36), and compiler information (GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)) on a darwin system. It also shows the help options: 'Type "help", "copyright", "credits" or "license" for more information.' The user then enters the command '>>> import this', which triggers the display of 'The Zen of Python, by Tim Peters'. The Zen of Python is a collection of 20 aphorisms about good programming practices, such as 'Beautiful is better than ugly' and 'Explicit is better than implicit'. The terminal ends with the prompt '>>>' and a cursor.

```

→ ~ python
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>

```

Figura 2.1: The Zen of Python

Main.java

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
import java.util.regex.*;

public class Main {

    public static void main(String[] args) throws IOException {
        PrintWriter textFile = null;
        try {
            textFile = new PrintWriter("result.txt");
            System.out.println("Enter the URL you wish to crawl..");
            System.out.print("@>");
            String targetUrl = new Scanner(System.in).nextLine();

            String response = getContentByUrl(targetUrl);

            Matcher matcher = Pattern.compile(
                "href=[\"'](?:[^\\"'"]+|\"\"|'')\"'")
            ).matcher(response);
            while (matcher.find()) {
                String url = matcher.group(1);
                System.out.println(url);
                textFile.println(url);
            }
        } finally {
            if (textFile != null) {
                textFile.close();
            }
        }
    }

    private static String getContentByUrl(String urlString)
        throws IOException {
        URL url = new URL(urlString);
        URLConnection urlConnection = url.openConnection();
        BufferedReader in = null;
        StringBuilder response = new StringBuilder();
        try {
            in = new BufferedReader(new InputStreamReader(
                urlConnection.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
        } finally {
            if (in != null) {
                in.close();
            }
        }
    }
}
```

```

    }
    }
    return response.toString();
}
}

```

Iar acum, varianta Python:

```

crawler.py

import re
import urllib.request

def main():
    with open("result.txt", "wt") as text_file:
        print("Enter the URL you wish to crawl..")
        url = input("@> ")
        print(url)
        for i in re.findall(
            "href=[\" '\"](.\" '\"]+)[\" '\"]",
            urllib.request.urlopen(url).read().decode(), re.I):
            print(i)
            text_file.write(i + '\n')

if __name__ == '__main__':
    main()

```

Ambele programe fac același lucru: primesc un URL de la utilizator, cer pagina aflată la acel URL și folosesc o expresie regulată pentru a găsi toate link-urile din pagina respectivă. Totuși, se poate observa că varianta scrisă în Python este mai scurtă și mai ușor de înțeles.

## 2.2 Django

*Django* este un framework web implementat în Python. Motto-ul lui este: "Un framework pentru perfecționiștii cu termene limită".

Django poate fi considerat un framework MVC, dar folosește o terminologie proprie:

- *Modelul* este reprezentat de ORM-ul (Object-relational mapping)<sup>3</sup> framework-ului care face legătura dintre obiecte Python și tabele dintr-o bază de date relațională.
- Clasele sau funcțiile care creează răspunsul la request sunt denumite *view*-uri. Acestea sunt denumite *controller*-ere în alte framework-uri MVC.
- Datele sunt prezentate folosind șabloane (templates). Acestea sunt ceea ce alte framework-uri numesc *view*-uri.
- *Controller*-ul este reprezentat de însuși framework, care rutează un request la view-ul corespunzător URL-ului cerut (prin URL dispatcher).

---

<sup>3</sup>[http://en.wikipedia.org/wiki/Object-relational\\_mapping](http://en.wikipedia.org/wiki/Object-relational_mapping)