Universitatea "BABEŞ-BOLYAI" Cluj-Napoca
Facultatea de Matematică şi Informatică
Departamentul de Informatică

Lucrare de Licență

Arhitecturi Orientate pe Servicii

Coordonator ştiinţific:
Prof. Dr. Bazil Parv

Absolvent:
Pop Radu Bogdan

Cluj-Napoca
2016

Cuprins

1. Introducere

1.1 Contextul proiectului

“A goal is a dream with a deadline.” Napoleon Hill

Într-o lume tot mai dinamică, procesul de evoluție este fundamentat şi generat de capacitatea fiecărui individ de a-şi folosi abilitățile spre a duce la bun sfârşit un angajament fixat în timp şi spațiu. Cu toții ştim că satisfacția de a termina orice lucru pe care ni-l propunem este una aparte şi esențială pentru a ne motiva încrederea în a progresa, dorindu-ne să îndeplinim sarcini tot mai dificile si mai provocatoare.

Analizând mai în detaliu acest proces primordial în dezvoltarea noastră ca oameni, constatăm că adesea suntem incerți în a ne defini cu exactitate un scop iar daca o facem, întâmpinăm dificultate atunci când încadrăm un scop în timp şi spațiu estimând de cele mai multe ori mult prea optimist finalitatea angajamentului nostru. Consecința unui potențial esec atunci când ne angajăm în a finaliza ceva este una definitorie cu un impact direct aspura psihicului nostru, riscând în multe situații să afectăm şi planurile altor oameni implicați direct sau indirect în activitatea noastră. Totodată acest proces în care ne definim un anume scop reprezintă un instrument perfect pentru a ne măsura capacitatea de a finaliza un target prin eficiența de care dăm dovadă. Monitrizând acest aspect este important să identificăm factorii cheie care ne încetinesc sau ne împiedică să nu realizăm ce ne-am propus.

1.2 Ideea şi scopul proiectului

Ideea proiectului “My personal development plan” este de a oferi potențialilor utilizatori oportunitatea de a folosi o aplicație complexă, prin care aceştia au la dispoziție un birou personal, menit să monitorizeze si să gestioneze într-o manieră analitică definirea întregului proces de dezvoltare personală a fiecarui utilizator.

Fie că vorbim din punct de vedere al dezvoltării personale, managementului resurselor umane ori managementul proiectelor, procesul de dezvoltare este consolidat dintr-o suită de scopuri definite de către utilizator în care sunt înglobate în detaliu o serie de obiective care marchează progresiv îndeplinirea scopului setat.

SMART este un acronim al caracteristicilor considerate esențiale în formularea scopurilor şi a obiectivelor:

„Ideal ar fi dacă fiecare corporație, departament şi secție ar avea obiective:

- ***S**pecifice*
- ***M**ăsurabile*
- ***A**ccesibile*
- ***R**ealistice*
- *încadrate în **T**imp* (1)

Specific – furnizează informații despre caracteristicile specifice unui anumit obiectiv. Obiectivul indică exact ceea ce se doreşte să se obtină si nu lasă loc de interpretări.

Măsurabil – prezintă aspecte cantitative si calitative ale unui obiectiv care pot fi măsurate cu unitațile de măsură cunoscute. Un obiectiv care nu are o modalitate de măsurare este ca un meci de fotbal în care nimeni nu ține scorul: toată lumea aleargă dar nimeni nu ştie cine a câstigat.

Accesibil – înseamnă că un obiectiv poate fi intr-adevăr atins cu capacitatea si resursele disponibile. Un obiectiv prin care vă propuneți să publicați trei cărți noi pe săptămână va pune o presiune extrem de mare pe echipa editorială şi vă duce la demotivarea acestora atunci când obiectivul nu va fi atins. O carte nouă la două săptămâni s-ar putea să se dovedească mai accesibil pentru resursele umane şi de timp ale unei edituri la un moment dat.

Relevant – înseamnă că realizarea obiectivului contribuie la impactul vizat de afacere. Realizarea unui obiectiv trebuie să contribuie în mod esențial la atingerea unui obiectiv mai mare, mai general. Relevanta obiectivului se evaluează în raport cu acest obiectiv mai general. In acest sens, el trebuie să vizeze un anumit impact.

Incadrat în Timp – face referire la un anumit interval de timp, bine precizat, privind stadiul atingerii obiectivului.

Cu o interfață foarte primitoare, aplicația îşi propune să pună în valoare toate aceste caracteristici care duc la parametrizarea eficace a obiectivelor şi a scopurilor, fiind practică atât utilizatorilor care îşi doresc să evolueze, coordonându-i să respecte aceste aspecte esențiale,

monitorizându-le perspicace progresul de dezvoltare cât şi pentru utilizatorii perfecţionişti care doresc să aibe o evidență clară asupra eficienței cu care progresează.

Funcționalitatea aplicației porneşte de la posibilitatea oricărui individ care deține un calculator sau un device conectat la internet de a-şi crea un cont pentru a avea acces la un birou personal. Odată creat acest cont, utilizatorul primeşte credențialele necesare pentru accesul deplin la folosirea instrumentelor accesibile biroului virtual.

Ca şi funcționalitate principală aplicația oferă posibilitatea ca utilizatorul să inregistreze scopuri. Fiecare scop poate să conțină mai multe obiective, utilizatorul având posibilitatea să asigneze oricâte obiective doreşte unui scop. De asemenea el poate să le modifice mai apoi sau să le şteargă. La crearea atât a unui obiectiv cât şi a unui scop, utilizatorul trebuie să estimeze efortul cuantificat în timp. Cuantumul efortului total ca şi timp al tuturor obiectivelor nu trebuie să depăşească efortul estimat pentru scopul din care fac parte obiectivele. Pe parcursul evoluției, utilizatorul poate să înregistreze ca şi timp progresul pe care l-a făcut urmând ca obiectivul să actualizeze timpul rămas dat de diferența dintre estimare şi timpii înregistrați.

Fiecare obiectiv şi scop are ca şi proces de evoluție urmatoarele stări:

1. Open
2. In progress
3. Done
4. Closed

In cazul în care un factor blochează evoluția stărilor se va putea opta pentru starea “Blocked”.

Este esential prioritizarea obiectivelor si a scopurilor, utilizatorul avand la dispozitie 4 indici: Critical, High, Medium, Low.

Aplicația mai pune la dispoziția utilzatorului o funcționalitate utilă şi rapidă de a-şi seta liste cu note scurte si precise. Notele sunt afişate în funcție de prioritatea setată. Biroul virtual dispune şi de un calendar interactiv în care se poate face planning la evenimente. Cu o structură de design foarte prietenoasă utilizatorul poate să-şi planifice evenimente în detaliu şi să le vizualizeze filtrat în alte cadre, dupa zi, săptămână sau lună. Ulterior aceste evenimente se pot modifica sau se pot şterge.

Partea cea mai interesantă si inovativă a aplicației este modulul analitic care aduce o serie de statistici grafice asupra obiectivelor şi a scopurilor definite de către utilizator. Această componentă este o radiografie perfectă şi cât se poate de exactă în ceea ce priveşte evoluția obiectivelor şi a scopurilor.

1.3 Structura proiectului

Intuitiv după tema aleasă, “Arhitecturi orientate pe servicii”, am dezvoltat o aplicație web independentă formată din doua module principale:

- Modulul grafic responsabil în interacțiunea cu utilizatorul numit şi client în termeni tehnici, proiectat să suporte cel puțin un serviciu de interconectare, care asigură logica efectivă a aplicației. Acest modul rulează independent şi este alocat pe un proces diferit față de restul aplicatiei.

- Modulul care asigură logica de business si procesarea/management-ul datelor pe care îl vom numi în cele ce urmează “backend as a service”.

Observăm o posibilă inconsistență între titlul lucrării “Arhitecturi orientate pe servicii” şi definirea conceptuală a aplicației în care practic avem un modul care joacă rolul de client şi doar un serviciu care susține cererile clientului. Intenționat am ales să proiectez în acest fel aplicația, tocmai ca la sfârşitul lucrării să fie deduse avantajele care asigură întradevar flexibilitatea de deschidere spre noi servicii înglobate odată cu dorința de dezvoltare a proiectului.

Pe de altă parte, această abordare pragmatică nu limitează sau afectează într-un fel serviciul construit responsabil în a asigura logica aplicației. Serviciul acesta va îndeplini toate caracteristicile pentru a fi considerat un API (Application Programming Interface) de sine stătător. Aşadar, arhitectura de proiectare a aplicației propusă face ca scop tema pe care o voi prezenta, lăsând la final o imagine clară asupra implementării acesteia.

1.4 Structura lucrării

În continuare, capitolele acestei lucrări sunt structurate astfel: aspecte teoretice ale proiectului, urmate de prezentarea la nivel general a unor considerente teoretice legate de partea de servicii din perspectiva web în vederea unei mai bune abstractizări a detaliilor ce țin de partea arhitecturala, tehnologiile folosite la implementarea proiectului, cu detalierea celor folosite la construirea modulelor.

Urmează arhitectura aplicaţiei, în care sunt prezentate diferitele module ale aplicaţiei şi interacţiunea dintre ele, din nou cu detalierea celor de backend direct implicate în schimbul de date între clienţi şi server.

Capitolul detaliilor de implementare prezintă metodele de programare folosite, oferind spre exemplificare porţiuni de cod explicate. În cadrul acestui capitol este descris mai în detaliu protocolul de comunicaţie între module, tipurile de url-uri folosite pentru accesul resurselor,

tipurile mesajelor schimbate şi diferitele tipuri de serializare. Mai apare şi detalierea interfaţării dintre server şi modulul de clusterizare.

În final, capitolul de utilizare a aplicaţiei prezintă aplicaţia şi modalitatea de utilizare a interfeţei grafice, prin screenshoturi şi înfăţişarea unor cadre de utilizare.

2. Prezentarea generală a serviciilor Web

2.1 World Wide Web

2.1.1 Aspecte generale

Spațiul WWW se poate definii pe larg ca fiind un sistem de distibuție locală sau globală a informațiilor de tip hipermedia, văzut ca şi un univers informațional compus din elemente de interes, deumite resurse, accesate de identificatiori globali – URI (Uniform Resource Identifiers).

Din perspectiva tehnică, Web-ul pune la dispoziție un sistem standardizat la nivel global de comunicare multimedia, informațiile fiind distibuite în funcție de cererile utilizatorilor având la bază modelul arhitectural client/server.

Metodele Web de stocare a informației în mod asociat, prin accesarea documentelor prin hiperlink-uri, si numirea site-urilor Web cu URL-uri fac din Web o extensie simplă a restului Internetului. Aceasta determină un acces facil la informație între diferitele servicii de Internet. Web-ul oferă, de asemenea, şi un spațiu unde companiile, organizațiile şi indivizii pot etala informații despre produsele, resursele si viețile lor, şi oricine care are acces la un computer conectat la Internet poate dispune de aceste informații – un mic procent al informațiilor de pe Web sunt disponibile numai utilizatorilor autorizați.

2.1.2 Istoric de evoluție

A fost dezvoltat de fizicianul şi informaticianul britanic Timothy Berners-Lee în cadrul Centrului European de Cercetare a Energiei Nucleare (CERN, actual Laboratorul European de Fizica Moleculara) din Geneva, Elvetia.

Prima implementare a devenit operationala CERN în 1989, şi s-a răspândit rapid prin intermediul instituțiilor de învățământ superior din domeniul fizicii moleculare. Comisia Europeană a aprobat primul proiect web (WISE) la sfârşitul anului 1991, încheind un parteneriat cu CERN.

Până în 1993 existau pe plan mondial 500 de servere web, WWW fiind “responsabil” de 1% din traficul pe Internet.

Anul 1994 a fost însa “Anul WWW”, în mai al aceluiaşi an având loc la sediul CERN prima Conferință Internațională World-Wide Web supranumită "Woodstock-ul Web-ului".

La sfârşitul anului 1994, Web-ul avea 10000 de servere, din care 2000 comerciale şi 10 milioane de utilizatori. Traficul era echivalent cu expedierea operelor complete ale lui William Shakespeare în fiecare secundă.

In ianuarie 1995, a fost fondat Consorțiul Internațional World-Wide Web "pentru a exploata World Wide Web-ul la maximul potențial".

2.2 Localizarea resurselor Web

Identificatorii uniformi de resurse – URI (Uniform Resource Identifiers) sunt secvențe alfanumerice şi reprezintă principala modalitate de adresare a resurselor Web. Mulțimea URI e compusă din nume uniforme URN (Uniform Resource Name) şi localizatori uniformi – URL (Uniform Resource Locator).

Adresele de tip URL sunt folosite în localizarea unei resurse prin protocolul HTTP. Adresele URL forma http://server:port/cale?interogare, unde în mod implicit, dacă nu este precizat, portul va avea valoarea 80, calea reprezintă un şir de directoare delimitate de "/", terminat eventual cu numele fisierului care stocheaza resursa adresată. Componenta interogare este opțională și desemnează un șir suplimentar, incluzând informații interpretate de resursă.

Limbajul HTML utilizează o sintaxă standard pentru exprimarea unui URL care are forma:

nume_serviciu://gazdă_Internet:număr_port/resursă

Avem 3 părţi distincte ale acestei sintaxe:

- numele serviciului (tipul resursei);
- gazda Internet (numele serverului) şi numărul portului necesar accesării resursei (uneori);
- numele fişierului (resursa).

Tipuri :

- URL-uri de tip http (hypertext transfer protocol):http://server/cale/fisier.html
- URL-uri de tip mailto: mailto:webmaster@www.adresa_server.ro
- URL-uri de tip ftp (file transfer protocol): ftp://server/cale/
- URL-uri de tip file: file://folder/cale/document1.doc

O adresa de tip URN desemnează un subset al URI care rămâne permanent și unic, chiar dacă resursa a dispărut ori a devenit inaccesibilă. URN-ul este utilizat pentru a desemna entități cum sunt de exemplu tipurile de date. Drept exemplificări, enumerăm:

- urn:mozilla:package:comunicator identifică pachete software ale suitei Mozilla;
- urn:scbeams-microsoft-com:datatypes desemnează tipurile de date definite de Microsoft;
- urn:ISBN:978-973-46-0249-0 desemnează numarul ISBN (International Standard Book number) asociat unei cărți.

2.3 Protocolul HTTP

Datorită importanței protocolului HTTP, consider utila o trecere în revistă a principalelor carateristici care stau la baza acestui protocol.

Hyper Text Transfer Protocol (HTTP) din perspectiva World Wide Web, este un protocol standardizat care facilitează la bază accesarea informațiilor în internet de pe serverele web. Fiind un protocol utilizat în internet, HTTP este regăsit la nivelul de aplicații al stivei de protocoale TCP/IP (Transmission Control Protocol/ Internet Protocol)

2.3.1 Versiuni

• HTTP/0.9 - prima versiune realizată de Tim Berners-Lee şi echipa sa. Această versiune este foarte simplă, dar cu numeroase neajunsuri, fiind repede înlocuită de alte versiuni;

• HTTP/1.0 – versiune introdusă în 1996 prin RFC1945, a adus numeroase îmbunătăţiri;

• HTTP/1.1 – versiune de îmbunătăţire şi reparare a neajunsurilor versiunilor anterioare. În prezent se utilizează două versiuni ale protocolului, HTTP/1.0 şi HTTP/1.1.

• La versiunea HTTP/1.0 se stabileşte o nouă conexiune TCP înaintea cererii, iar după transmiterea răspunsului conexiunea trebuie închisă. Astfel dacă un document HTML cuprinde 10 imagini, vor fi necesare 11 conexiuni TCP, pentru ca pagina să fie afişată complet (în browser).

• La versiunea 1.1 se pot emite mai multe cereri şi răspunsuri pe aceeaşi conexiune TCP. Astfel pentru documentul HTML cu 10 imagini este necesară doar o singură conexiune TCP. Datorită algoritmului Slow-Start - viteza conexiunii TCP este la început mică, dar acum el e necesar doar

o singură dată, se scurtează semnificativ durata totală de încărcare a paginii. La aceasta se adaugă şi faptul că versiunea 1.1 poate relua şi continua transferuri întrerupte.[1]

2.3.2 Modul de funcționare

Pentru motivația practică a urmatoarelor subcapitole vom porni de la urmatorul exemplu:

Odată accesat un link sau o adresă de web cum ar fi http://adresa-server, atunci se cere calculatorului host să afişeze o pagină web (index.html sau altele). În prima fază numele (adresa) adresa-server este convertit de protocolul DNS într-o adresă IP. Urmează transferul prin protocolul TCP pe portul standard 80 al serverului HTTP, ca răspuns la cererea HTTP-GET. Informaţii suplimentare ca de ex. indicaţii pentru browser, limba dorită ş.a. se pot adăuga în header-ul (antetul) pachetului HTTP. În urma cererii HTTP-GET urmează din partea serverului răspunsul cu datele cerute, ca de ex.: pagini în (X)HTML, cu fişiere ataşate ca imagini, fişiere de stil (CSS), scripturi (Javascript), dar pot fi şi pagini generate dinamic (SSI, JSP, PHP şi ASP.NET).

Dacă dintr-un anumit motiv informaţiile nu pot fi transmise, atunci serverul trimite înapoi un mesaj de eroare. Modul exact de desfăşurare a acestei acţiuni (cerere şi răspuns) este stabilit în specificaţiile HTTP.

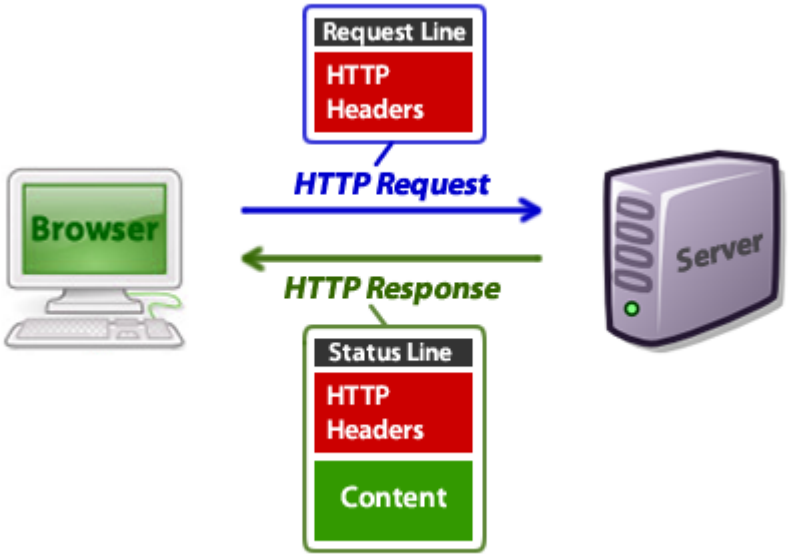


Fig.1 Diagrama generala de interactiune a serviciu HTTP [2]

[1] R. Fielding, J. G.-L. (1999, June). *Hypertext Transfer Protocol -- HTTP/1.1*.
[2] Saenz, J. (n.d.). Building Web Apps with Go.

Deseori utilizatorul doreşte să transmită informaţii speciale la website. Aici HTTP pune la dispozitie două posibilităţi:

- Transferul datelor în combinaţie cu o cerere pentru o resursă (HTTP-metoda "GET")
- Transferul datelor în combinaţie cu o cerere specială (HTTP-metoda "POST").

Pentru întelegerea deplină a exemplului propus, am urmărit detalierea noțiunilor generale ce țin de protocolul HTTP în cele ce urmează.

2.3.3 Concepte fundamentale

La nivel structural, elementele care stau la baza sunt *cererea* şi *răspunsul*: un client Web trimite un mesaj (cerere) la un server. Mesajul este compus din identificatorul resursei dorite specificat sub forma unui URI (Unifor Resource Identifier), metoda acces folosită, versiunea protocolului precum şi o serie de meta-informații care pot fi utile serverului. Raspunsul serverului cuprinde un cod ce reflectă starea serverului după procesarea cererii, un mesaj explicativ pentru codul de stare transmis, meta-informațiile care vor fi procesate de către client şi, poisbil, un conținut cum ar fi resursa solicitată. Dacă dintr-un anumit motiv informațiile nu pot fi transmise, atunci serverul trimite înapoi un mesaj de eroare. Modul exact de desfășurare a acestei acțiuni (cerere și răspuns) este stabilit în specificațiile HTTP.

Un concept important este cel de caching. Un cach reprezentă un depozit local folosit pentru stocarea datelor la nivel de server/client. O sesiune de comunicare HTTP este inițiată de către un un client si are ca mecanism de bază solicitarea unei resurse identificată unic pe un server numit siserver de origine.

În procesul de comunicare dintre client si server pot să apară unul sau mai mulți intermediari cum ar fi: proxy (numit si server proxy), poartă (gateway) sau tunel (tunnel):

- Un server proxy are ca şi scop preluarea unei cereri si transmiterea ei mai departe eventual modificată către server. Orice proxy deține obligatoriu un cache, regăsit sub denumirea de sistem de cache
- Poarta semnifică un intermediar amplasat înaintea unui server de origine care se poate identifica drept acesta, clientul web necunoscând acest aspect.
- Tunelul joacă rolul tot al unui intermediar care nu schimbă însă conținutul mesajului, ci are rol exclusiv de retransmitere a lui.

Mai jos putem vedea exemplul tipic de cerere-răspuns folosit în aplicația My Personal Development Plan care poate să conțină formatul următor:

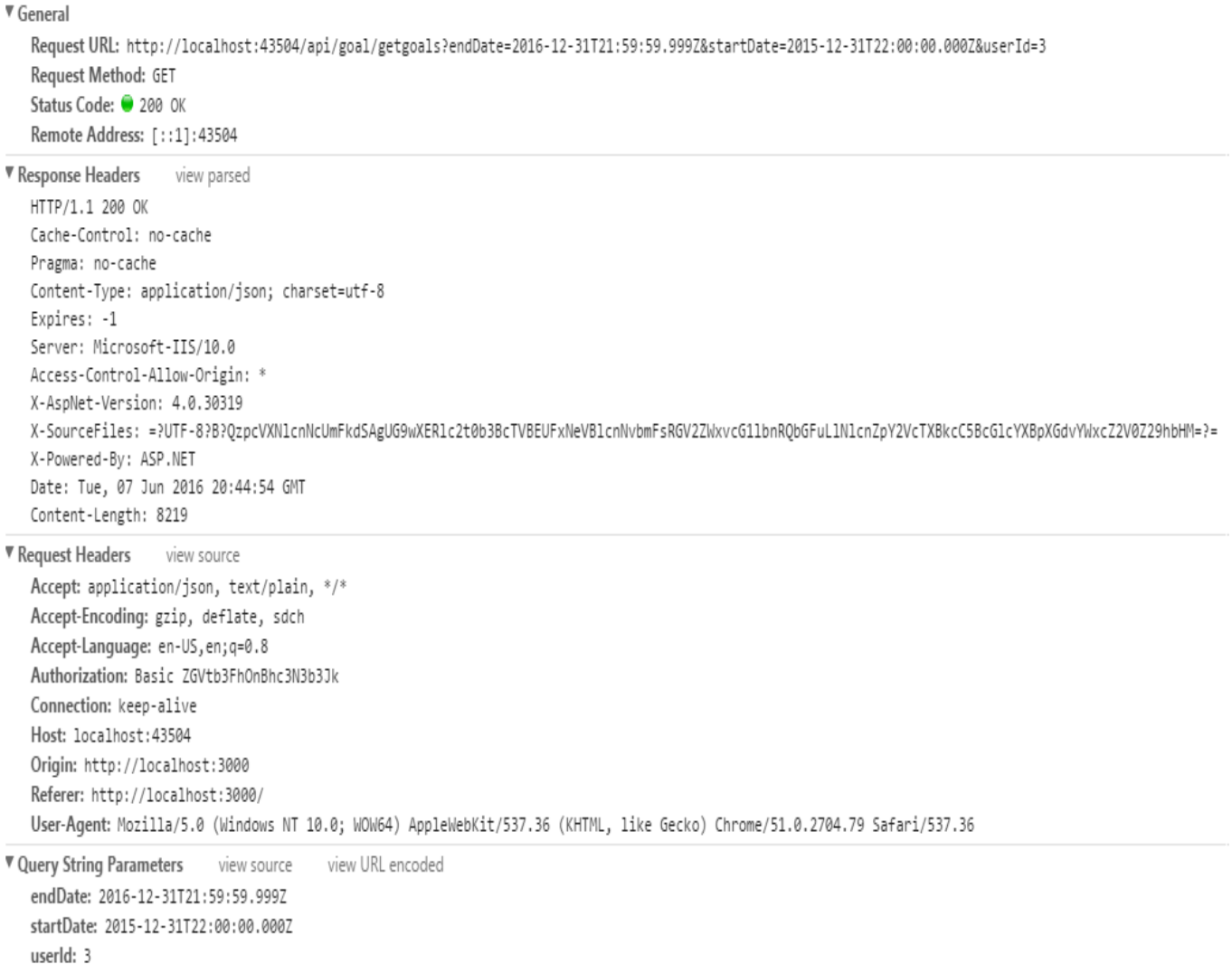


Fig. **2** Exemplu de cerere/raspuns in aplicatia My Personal Development Plan

Corpul unui posibil răspuns obținut din partea API-ului care joacă rol de Server As Service.


```
▼{goals: [{Id: 1, UserProfileId: 3, Username: "demoqa", Name: "Goal updated 11",…},…], goalsCount: 12}
  ▼goals: [{Id: 1, UserProfileId: 3, Username: "demoqa", Name: "Goal updated 11",…},…]
    ▼0: {Id: 1, UserProfileId: 3, Username: "demoqa", Name: "Goal updated 11",…}
       DateCreated: "2016-05-28T02:28:17.297Z"
       Description: "No desc"
       Estimation: "P1D"
       GoalStatus: "InProgress"
       Id: 1
       Name: "Goal updated 11"
      ▼Objectives: [{Id: 5, GoalId: 1, Title: "Take a break", Description: "Holiday", Progress: 0,…},…]
        ▶0: {Id: 5, GoalId: 1, Title: "Take a break", Description: "Holiday", Progress: 0,…}
        ▶1: {Id: 11, GoalId: 1, Title: "Objective 11", Description: "No description", Progress: 0,…}
        ▶2: {Id: 12, GoalId: 1, Title: "Smart objective", Description: "Description", Progress: 0,…}
        ▶3: {Id: 14, GoalId: 1, Title: "Objective d", Description: "Description", Progress: 0,…}
       ObjectivesCount: 4
       Progress: 50
       RemainingEstimates: "P1D"
      ▼UserProfile: {Id: 3, Name: "Qa Demo", Username: "demoqa", Email: "qa@qa.com", Location: null}
          Email: "qa@qa.com"
          Id: 3
          Location: null
          Name: "Qa Demo"
          Username: "demoqa"
       UserProfileId: 3
       Username: "demoqa"
    ▶1: {Id: 2, UserProfileId: 3, Username: "demoqa", Name: "Second goal",…}
    ▶2: {Id: 3, UserProfileId: 3, Username: "demoqa", Name: "My own goal",…}
    ▶3: {Id: 4, UserProfileId: 3, Username: "demoqa", Name: "Holiday", DateCreated: "2016-05-30T02:51:07.627Z",…}
    ▶4: {Id: 5, UserProfileId: 3, Username: "demoqa", Name: "Getting done with project a",…}
    ▶5: {Id: 6, UserProfileId: 3, Username: "demoqa", Name: "Project b",…}
    ▶6: {Id: 7, UserProfileId: 3, Username: "demoqa", Name: "Project 10",…}
    ▶7: {Id: 8, UserProfileId: 3, Username: "demoqa", Name: "Lazar goals",…}
    ▶8: {Id: 9, UserProfileId: 3, Username: "demoqa", Name: "Goal example",…}
    ▶9: {Id: 10, UserProfileId: 3, Username: "demoqa", Name: "Goal example2",…}
    ▶10: {Id: 11, UserProfileId: 3, Username: "demoqa", Name: "Goal test",…}
    ▶11: {Id: 12, UserProfileId: 3, Username: "demoqa", Name: "Miha's goal",…}
    goalsCount: 12
```


Fig. 3 Corpul răspusului inițiat în aplicația My Personal Development Plan

2.3.4 Mesajele HTTP

Putem clasifica cererile și răspunsurile HTTP ca fiind mesaje direcționate, acestea pot fi considerate de doua tipuri: cerere provenită de la un client către un server și răspuns al serverului trimis către clientul care l-a solicitat. Un mesaj HTTP contine o succesiune de linii de text, delimitate de CLRF (Carriage Return Line Feed). Prima linie este ocupată de cererea efectuată de către client sau de un cod de stare obținut de la server, urmată de un număr de atribute de antet.

Din punct de vedere structural un mesaj conține o secțiune numită antent (header) care are in compoziție mai multe atribute folosite la completarea cererii sau a unui răspuns cu meta-informația necesară interpretării corecte a mesajului prin stabilirea unor valori specificate de către protocolul HTTP sau a unor protocoale definite cum este SOAP. Meta informatiile (meta-data) sunt date care descriu specific alte date în vederea procesării lor mai departe. Meta informațiile se regăsesc şi în fişierele HTML. Un atribut este furnizat printr-un nume urmat de ":" și de o valoare.

O cerere este specificată de o metodă de acces. Un posibil exemplu de cerere este urmatorul:

GET specificare_resursa HTTP/1.1 CRLF

Forma generala a unui mesaj de cerere este conforma schemei de mai sus:

Metoda | resursa | versiune_protocol CRLF

Versiunea de protocol trebuie specificată deoarece nu toate serverele au implementat ultima versiune sau nu toți clienții o cunosc. Deci, pentru ca totuşi un server "deştept" să se poată întelege şi cu un client mai putin dotat, sau invers, şi fără a renunța la posibilitățile introduse de versiunile (mereu mai) noi ale protocolului, trebuie să se realizeze mai întâi o negociere între server si client, relativ la ce ştie fiecare şi abia apoi să se desfăşoare transferul propriu-zis de date.

Metodele sunt de fapt operațiile care pot fi aplicate obiectelor constituite de resursele din rețea, în accepțiunea protocolului HTTP. Metoda va trebui să fie totdeauna primul element dintr-o linie de cerere. Metodele prevăzute în versiunea 1.1 sunt următoarele: OPTIONS, GET, HEAO, POST, PUT, PATCH, COPY, MOVE, DELETE, LINK, UNLINK, TRACE, WRAPPED.

OPTIONS semnifică o cerere relativă la informațiile ce definesc opțiunile de comunicare disponibile pe conexiunea către URI-ul specificat în cerere. Metoda permite determinarea opțiunilor şi/sau posibilităților unui server, fără să determine o acțiune din partea resursei adresate.

Metoda are nevoie de parametrii, nu numai resursă, iar în HTTP termenul consacrat pentru parametrii metodelor este "***header field***" sau "***antet de cîmp***". Definite în cadrul protocolului pentru fiecare metodă, antetele de câmp pot avea valori care la rândul lor sunt definite (dar nu limitate, extensiile fiind în principiu totdeauna posibile).

Exemplu:

O cerere de tipul

OPTIONS www.abc.ro HTTP1/1 CRLF Accept: audio/*; q=0.2, audio/basic CRLF

reprezintă o cerere de definire a opțiunilor către serverul www.abc.ro, în care clientul solicitant spune că preferă audio/basic, dar acceptă orice tip pentru date audio în cazul în care calitatea reprezentarii nu scade sub 20%.

Exemplu:

O cerere de genul:

OPTIONS www.abc.ro HTTP1/1 CRLF Accept: text/plain; q=0.5, text/html, text/x-dvi;q=0.8; mxb=100000, text/x-c CRLF

specifică urmatoarele preferințe relative la modul de reprezentare al textului: x-c sau html, dacă sunt disponibile; daca nu, x-dvi, dar numai daca textul nu depaşeşte 100000 de octeți, sau plain.

Virgula separă opțiunile posibile, punct-virgula separă determinările sau preferințele suplimentare relative la o anumită opțiune.

GET este una dintre cele mai importante metode si singura care era disponibilă în prima versiune a protocolului, HTTP/0.9. GET este metoda care "aduce" ceva de la resursă; mai concret, dacă resursa este un proces care produce date (o cautare de pildă), răspunsul la metoda GET va fi o entitate care să cuprindă acele date. Răspunsul este unul singur: aceasta este o caracteristică de bază a protocolului. Chiar dacă volumul de date care trebuie incluse în răspuns este mare, nu se face o fracționare în bucatele mai mici, care să permită transferul mai usor al răspunsului. Din punct de vedere al protocolului HTTP, discuția este totdeauna simplă: o întrebare are un răspuns. Nu se pot pune mai multe întrebări pentru a obtine un singur răspuns, nu se pot formula mai multe răspunsuri la o întrebare.

Există totuşi două posibilittți de a micşora volumul de date care să circule pe rețea în urma elaborării unui răspuns; o condiționare de genul "dacă s-a schimbat ceva" şi posibilitatea de a prelua numai o parte din acesta. De exemplu, o cerere de genul:

GET www.abc.ro/?cerere HTTP/1.1 If-Modified-Since: Wed, 24 Mar 1999 1:00:00 GMT

va aduce ceea ce s-a cerut numai dacă s-a modificat ceva după data şi ora specificate în parametrii metodei.

HEAD este o metodă similară cu GET, folosită în principiu pentru testarea validității şi/sau accesibilității unei resurse, sau pentru a afla dacă s-a schimbat ceva. Sintaxa este similară metodei GET; spre deosebire de GET însă, datele eventual produse de resursă în urma cererii nu sunt transmise; doar caracteristicile acestora, şi un cod de succes sau eroare. Ceva de genul "dacă ți-as cere să execuți cererea mea, ce mi-ai răspunde?".

POST este metoda prin care resursei specificate în cerere i se cere să își subordoneze datele incluse în entitatea care trebuie să însoțească cererea. Cu POST se poate adauga un fisier unui anumit director, se poate trimite un mesaj prin poşta electronică, se poate adăuga un mesaj unui grup de ştiri, se pot adăuga date unei baze de date existente, etc. Metoda POST este generală; care sunt procesele pe care un anumit server le acceptă sau cunoaşte îi sunt strict specifice.

PUT este o metodă care cere serverului ceva mai mult decât POST: să stocheze/memoreze entitatea cuprinsă în cerere cu numele specificat în URI. Dacă resursa specificată există deja, entitatea nouă trebuie privită ca o versiune modificată care ar trebui să o înlocuiască pe cea existentă. Serverul, bineânțeles, va accepta sau nu această cerere, funcție de drepturile de acces pe care i le-a acordat clientului, şi va răspunde cererii cu informații corespunzătoare ("s-a făcut", "nu pot", "nu ai voie să faci treaba asta" etc.). Pentru a evita situații care să ducă la încărcarea excesivă şi nejustificată a rețelei - de exemplu, un client care vrea să "posteze" un text de 10 MB, fară să țină seama de faptul că serverul nu mai are atât loc atât o cerere de tipul POST cât şi una de tipul PUT se desfaşoara în doi timpi: întăi, clientul trimite numai parametrii metodei, fără să trimită datele efective pe care le vrea postate după care asteaptă 5 secunde. În acest timp, dacă serverul răspunde, clientul ia în seamă şi analizează răspunsul serverului (iar daca acesta este "nu mai am loc", datele nu se mai transmit). Dacă nu soseşte nici un răspuns în timpul de aşteptare, se consideră implicit că serverul acceptă datele şi acestea sunt transmise de către client.

PATCH este o metodă similară lui PUT, dar nu conține toate datele care să definească resursa, ci numai diferențele față de versiunea existentă pe server, cu toate informațiile necesare care să îi permită serverului să reconstruiască o versiune la zi a resursei.

COPY, MOVE şi **DELETE** sunt metode prin care se cere ca resursa specificată în URI-ul din cerere să fie copiată în locațiile specificate ca parametri pentru metoda, mutată acolo sau respectiv doar ştearsă.

LINK şi **UNLINK** sunt metode prin care resursa specificată în cerere este legată/dezlegată de alte resurse, stabilind una sau mai multe relații cu acestea din urmă. Specificate ca parametrii pentru metodă ar putea fi de exemplu un index pentru o baza de date, un cuprins pentru un set de documente, etc.

TRACE este o metodă care îi permite clientului să vadă cum ajung cererile sale la server, pentru a verifica/diagnostica conexiunea, pentru a se verifica pe sine sau pentru a determina felul în care

eventualele proxy-uri de pe parcurs au modificat cererea inițiala. Serverul, în raspuns la această cerere, va trimite în ecou cererile care îi vin de la client, fără să le mai trateze ca cereri "reale".

WRAPPED este o metodă care "contrazice" principiul protocolului de a trimite totdeauna o singură cerere şi a aştepta un singur răspuns. Via WRAPPED, mai multe cereri, care în mod obişnuit ar fi succesive, sunt "împachetate" într-una singura. Iar o altă aplicare a metodei țintește măsuri de securizare - o cerere poate fi cifrată şi transmisă prin metoda WRAPPED, ceea ce va determina serverul să acționeze în doi paşi: întîi să descifreze cererea reală, iar apoi să îi dea curs acesteia.

În cadrul unei cereri pot fi specificate diverse atribute, utilizate pentru a transmite serverului informații suplimentare privitoare la acea cerere și la client. După primirea și interpretarea unei cereri, serverul întoarce un răspuns. Prima linie conține așa cum am mai precizat, versiunea protocolului HTTP implementat de către server si un cod de stare format dintr-un număr compus din trei cifre care semnifică felul in care a fost procesată starea. După acest numar urmează un text explicativ pentru codul de stare returnat.

Codul de stare e format din trei cifre organizate în categorii de stări. Ele se disting dupã prima cifrã în modul următor:

- Coduri de informare (1xx) – furnizează informații cu privire la o anumită acțiune (cererea a fost primită, comunicația continuă). De exemplu: 100 Continue (clientul poate continua cererea fiind nevoit să trimită următoarea parte a unui mesaj parțial). Un alt exemplu este 102 Processing – Acest cod indică faptul că serverul a primit şi prelucrează cererea, dar nici un răspuns nu este disponibil încă. Acest lucru îl ajuta pe utilizator să nu închidă cererea presupunând că informația a fost pierdută.

- Coduri de succes (2xx) – indică efectuarea cu succes a unei operațiuni (cererea a fost primită, interpretată şi acceptată de către server). Un cod tipic pentru această categorie este 200 OK. Alt exemplu este 204 No Content (serverul a rezolvat cererea, dar nu returnează nimic clientului).

- Coduri de redirecționare (3xx) – indică o redirecționare a cererii spre altă locație ori alt server. Ca si exemple putem menționa codurile 301 Moved Permanently (resursa solicitată a fost asociată unui URI nou și orice referință viitoare la ea trebuie să se realizeze prin acest URI furnizat) și 302 Moved Temporarily (resursa cerută are asociat un alt URI, dar pentru o perioadă temporară).

- Coduri de eroare provocate de client (4xx) - specifică apariția unei erori pe partea clientului (fie cererea este incorectã din punct de vedere sintactic, fie nu poate fi satisfăcută din diverse motive). Ca exemple, returnăm 401 Unauthorized (cererea necesitã autentificarea utilizatorului.

- Coduri de eroare generate de server (5xx) desemnează coduri semnificând o eroare pe partea serverului (cererea este aparent corectã, dar serverul nu o poate îndeplini din anumite motive). Drept exemplificare putem menționa 503 Service Unavailable.

Tabelul de mai jos ilustrează atribute cele mai des întalnite care inoștesc mesajele HTTP.

Atribut HTTP	***Utilizare***
Accept	Specific unei cereri, cu rol în stabilirea tipului conținutului prin intermediul unei negocieri conduse de server; clientul are posibilitatea de a furniza tipurile media (MIME) pe care acesta le recunoaște și le poate interpreta sau poate indica numai tipul de răspunsuri preferate. Un exemplu este Accept: text/xml, application/xml
Host	Folosit într-o cerere pentru specificarea adresei Internet și a portului în vederea stabilirii exacte a locației unde se gasește resursa căreia i se adresează cererea Host: www.abc.com
Cache-Control	Permite controlul cache-ului, de cele mai multe ori la nivelul proxy-ului dintre client și server Cache-Control: max-age=600
Connection	Atribut general folosit pentru a specifica anumite proprietăți legate de conexiune. Se aplică doar comunicaþiei între două aplicaþii HTTP din lanțul unor cereri sau răspunsuri. E util în implementarea conexiunilor persistente Connection: close
Content-Type	Desemnează tipul MIME al reprezentării resursei solicitate de un client ori transmise de server. Un exemplu notoriu este Content-Type: text/html
Location	Specific răspunsurilor HTTP. Utilizat în conjuncție cu coduri de stare de tip 3xx sau 201 Created. Poate stabili locația curentă sau preferată a resursei sub forma unui URI absolut.
Server	Conține informații despre aplicația server, cum ar fi numele, versiunea, producătorul, aplicațiile compatibile Server: libwww-perl-daemon/1.36

2.4 Protocolul HTTPS

Secure Hyper Text Transfer Protocol (sau HyperText Transfer Protocol/Secure, abreviat HTTPS) reprezintă protocolul HTTP încapsulat într-un flux SSL/TLS care criptează datele transmise de la un browser web la un server web, cu scopul de a se oferi o identificare criptată și sigură la server.

Conexiunile HTTPS sunt folosite în mare parte pentru efectuarea de operațiuni de plată pe World Wide Web și pentru operațiunile "sensibile" din sistemele de informații corporative. HTTPS nu trebuie confundat cu Secure HTTP (S-HTTP) specificat în RFC 2660.

HTTPS este un protocol de comunicație destinat transferului de informație criptată prin intermediul WWW. A fost dezvoltat din necesitatea de a proteja de intruși transferul datelor prin HTTP - un protocol "clear-text", prin care datele de pe server-ul web sunt transmise browser-ului client în clar, posibilitățile de a intercepta acest transfer constituind tot atâtea posibilități de a accesa și utiliza fără restricții informațiile respective.

HTTPS nu este altceva decât HTTP "încapsulat" cu ajutorul unui flux SSL/TLS - datele sunt criptate la server înainte de a fi trimise clientului, astfel încât simpla interceptare a acestora pe traseu să nu mai fie suficientă pentru a avea acces la informații.

HTTPS este în același timp o metodă de autentificare a server-ului web care îl folosește, prin intermediul așa-numitelor "certificate digitale" - o colecție de date pe care un browser o solicită server-ului pentru a putea începe transferul criptat; dacă certificatul este emis de o autoritate cunoscută (de exemplu VeriSign), browser-ul poate fi sigur că server-ul cu care comunică este ceea ce pretinde a fi.

3. Arhitectura orientată pe servicii Web

3.1 Scurtă introducere

O aplicație web se poate defini ca fiind un sistem software bazat pe o arhitectură de tip client server care folosește tehnologiile deschise World Wide Web.

Din punct de vedere al proiectării software-ului, dezvoltarea aplicațiilor web este un nou domeniu al aplicațiilor[3]. În ciuda anumitor similitudini cu aplicațiile tradiționale, caracteristicile speciale ale aplicațiilor web necesită o adaptare a multiplelor abordări ale proiectarii software-lui sau chiar a dezvoltării de abordări complet noi.

Principiile de bază ale proiectării web pot fi descrise în mod similar cu cele ale proiectarii software:

- obiective şi cerințe clar definite;

- dezvoltarea sistematică, în faze, a aplicațiilor web;

- o planificare foarte atentă a acestor faze;

- auditul continuu a întregului proces de dezvoltare.

Proiectarea web face posibilă planificarea si repetarea proceselor de dezvoltare şi în acest mod facilitează evoluția continua a aplicațiilor web. Aceasta permite nu doar reducerea costurilor şi minimizarea riscului pe parcursul dezvoltării şi întreținerii, ci şi creşterea calității, precum şi măsurarea calității rezultatelor fiecărei faze[4].

Din punctul de vedere al vizibilității, o aplicație Web poate fi disponibilă doar în cadrul unui intranet (rețeaua internã proprie unei companii sau organizații) și/sau în extranet (extindere a facilităților intranetului prin mijlocirea comunicațiilor între intraneturile a două sau mai multe organizații).

3.2 Servicii web

Originile și scopurile Web-ului iau în considerare constituirea unui spațiu de comunicare interumanã prin intermediul partajării cunoştințelor şi exploatarea puterii computaționale puse la

[3] Glass, R. L. (2003). A mugwump's-eye view of Web work. *Communications of the ACM*, Pages 21-23.
[4] Mendes, E. M. (2006). *Web Engineering.*

dispoziție de calculatoarele interconectate.[5] Pornind de la această premisă, o prima abordare este aceea de a recurge la un mecanism care sa ne permită apelarea unor operații în vederea executarii lor de la distanță. Astfel, un program client este apt sa invoce functionlitați localizate pe alte calculatoare din rețea.

Un serviciu Web (Web Service) este o aplicație Web de tip client-server, în care un server furnizor de servicii (numit şi "Service Endpoint") este accesibil unor aplicații client pe baza adresei URL a serviciului. Putem defini generic un serviciu web ca fiind o componentă software care poate fi accesată folosind la baza protocolul standard de retea HTTP. Intrebarea *"Cum poate fi accessat un serviciu web?"* este cea mai bună premisă pentru a descrie subcapitolul *3.2.1 Clasificarea serviciilor web*. Serviviciile Web furnizează un standard ce facilitează interoperabilitatea aplicațiilor software ce rulează pe o varietate de platforme și framework-uri. Furnizorul de servicii expune un API (Application Programming Interface) pe Internet, adicã o serie de metode ce pot fi apelate de clienti. Aplicatia client trebuie sã cunoascã adresa URL a furnizorului de servicii si metodele prin care are acces la serviciul oferit. Voi detalia intr-un capitol urmator aspecte si particularitați ce tin de API.

3.2.1 Clasificarea serviciilor web

Diferența generală dintre o aplicatie Web clasicã şi un serviciu Web constã în principal în formatul documentelor primite de client şi a modului cum sunt ele folosite : într-o aplicatie Web clientul primeşte documente HTML transformate de un browser în pagini afişate, iar clientul unui serviciu Web primeşte un document XML (sau JSON) folosit de aplicația client, dar care nu se afişazã direct (decât în anumite programe de verificare a serviciilor Web).

Din punct de vedere al tehnologiilor folosite existã douã tipuri de servicii Web:

- Servicii de tip **REST** (RESTful Web Services), în care cererile de la client se exprimã prin comenzi HTTP (GET, PUT, POST,DELETE), iar rãspunsurile sunt primite ca documente XML sau JSON;

- Servicii de tip **SOAP** (Simple Object Access Protocol), în care cererile şi rãspunsurile au forma unor mesaje SOAP (documente XML cu un anumit format) transmise tot peste HTTP. In astfel de servicii furnizorul expune şi o descriere a interfeței API sub forma unui document WSDL (Web Service Description Language), care este tot XML şi poate fi prelucrat de client. Un client trebuie sã cunoascã metodele oferite de cãtre "Service Endpoint", pe care le poate afla din descrierea WSDL.

[5] Lenuța Alboaie, S. B. (2006). *Servicii Web - Concepte de baza şi implementari.* POLIROM.

Serviciile de tip SOAP oferã mai multã flexibilitate, o mai bunã calitate a serviciilor şi interoperabilitate fata de serviciile REST şi sunt recomandate pentru un API mai mare oferit clientilor. Serviciile de tip SOAP pot fi combinate pentru realizarea de operații complexe, ceea ce a condus la o arhitecturã orientatã pe servicii (SOA=Service Oriented Architecture).

Majoritatea serviciilor Web reale sunt de tip SOAP, dar existã si câteva servicii de tip REST notabile : Amazon S3 (Simple Storage Service) pentru memorarea si regãsirea de obiecte, reteaua Twitter si alte site-uri de blog, în care se descarcã fişiere XML în format RSS sau Atom cu liste de legãturi cãtre alte resurse. Ca exemple de servicii SOAP larg utilizate sunt servicii pentru rate de schimb între diferite valute, servicii bancare pentru verificare si operare în conturi (de cãtre aplicații Web de comerț electronic de exemplu), servicii de memorare si regãsire date, etc.

3.3 Arhitectura RPC (SOAP)

Un serviciu Web de tip SOAP este versiunea modernã a unui apel de proceduri la distantã (numite RPC=Remote Procedure Call sau RMI=Remote Method Invocation), cu diferența cã datele schimbate între client şi server sunt documente XML în locul fisierelor binare care respectau si ele anumite conventii (protocolul RMI-IIOP sau CORBA sau alt protocol). [6]

RPC este o tehnica puternică pentru construirea aplicațiilor distribuite bazate pe modelul client-server. Modelul extinde noțiunea de apel local de procedură, diferența fiind ca procedura apelată nu se afla in acelasi spațiu de adresare cu procedura apelantă. Cele doua procese implicate pot sa fie pe același calculator sau pot să fie pe două calculatoare în rețea. Utilizând PRC, programatorii de aplicații distribuite ocolesc dezvoltarea interfațării aplicației cu reteaua.

Protocolul RPC se află la nivelul Prezentare din stiva OSI. RPC foloseşte protocolul XDR (eXternal Data Representation) – RFC 1832 - pentru codarea datelor. Acest protocol se află tot la nivelul Prezentare din stiva OSI. Versiunea originală de RPC a fost definită in RFC 1050. Versiunea 2 de RPC e definită în RFC 1057, iar versiunea ONC-RPC (Open Network Computing. Versiunea 2 e definită în RFC 1831. Protocolul SOAP este astãzi unanim folosit în aplicații de tip servicii Web şi înlocuieşte protocoale mai vechi de comunicare între obiecte aflate pe maşini diferite ca IIOP(RMI), CORBA, DCOM, JAX-RPC s.a.

Un mesaj SOAP este un document XML cu o anumitã structurã. Mesajele SOAP au o parte care nu poate lipsi (SOAP Part) şi o parte ataşatã, opțională (Attachements).

[6] Suda, B. (2003). *SOAP Web Services.*

Partea fixã contine o anvelopã (SOAP Envelope), anvelopã care conține o parte optionalã de antet (SOAP Header) şi o parte obligatorie de corp mesaj (SOAP Body) :

SOAP message
SOAP part
SOAP envelope
SOAP header (optional)
Header
Header
SOAP body
XMLContent
SOAP Fault

Fiecare din pãrtile ataşate conține o parte de antete MIME şi o parte de continut (XML sau non-XML):

O aplicație RPC va consta dintr-un client şi un server, serverul fiind localizat pe maşina care executã procedura. Aplicația client comunicã prin reȩea via TCP/IP cu procedura de pe calculatorul aflat la distanță, transmițând argumentele şi recepționând rezultatele. Clientul şi serverul se executã ca douã procese separate care pot rula pe calculatoare diferite din rețea acestui port.

Remote Procedure Call (Apelul Procedurilor la Distanta) este o tehnologie care pemite unui software să apeleze o altă procedură (subrutina, funcție) pe altă maşină (in mod uzual pe alt computer), fără ca programatorul să fie nevoit să cunoască toate detaliile în care cele două sisteme interacționează. Programatorul va scrie acelaşi cod indiferent dacă va apela subrutina local sau remote.

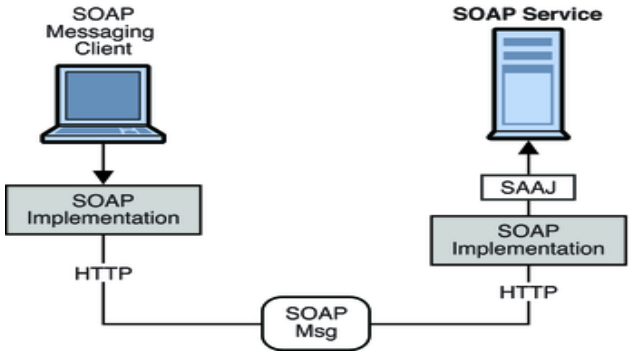


Fig. 4 Diagrama generala a serviciului SOAP[7]

[7] Oracle. (2011, July). *Oracle GlassFish Server Message Queue 4.5 Developer's Guide for Java Clients*. Retrieved from Oracle.

3.4 Arhitectura REST

3.4.1 Noțiuni generale

Representational State Transfer (REST) reprezintă un stil arhitectural bazat pe standardele web şi pe protocolul HTTP, pentru sisteme distribuite că World Wide Web. REST s-a evidenţiat în ultimii ani ca modelul de design predominant al web-ului, datorită simplităţii sale. REST a fost descris în 2000 de Roy Fielding, în lucrarea sa de doctorat.[8]

REST se referă strict la o colecţie de principii arhitecturale într-o reţea care subliniază felul în care resursele sunt definite şi adresate. Într-un context mai puţin riguros, REST descrie orice interfaţă care transmite date prin HTTP fără un nivel adiţional de mesagerie cum ar fi SOA sau folosirea sesiunilor prin cookie-uri HTTP. Este posibil să dezvoltăm sisteme software în concordanţă cu stilul arhitectural REST fără să folosim HTTP şi fără să interacţionăm pe World Wide Web. De asemenea, este posibil să dezvoltăm un sistem software care să folosească HTTP şi XML care să nu respecte principiile REST, urmând în schimb modelul arhitectural RPC (remote procedure call).

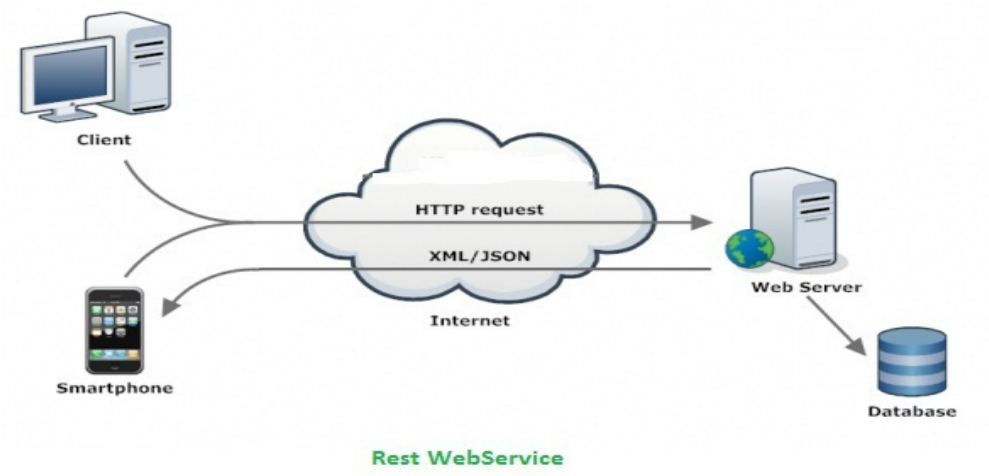


Figure 5 Diagrama de interactiune a unui serviciu REST[9]

În serviciile web care se bazează pe arhitectura REST, informaţiile despre metoda apelată sunt date de metoda HTTP folosită, iar argumentele metodei sunt date de URI-ul folosit. Combinaţia este puternică, astfel din prima linie a unei cereri HTTP făcută unui serviciu web în manieră REST (”GET /reports/open-bugs HTTP/1.1”) ar trebui să înţelegem ce doreşte clientul

[8] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Preluat de pe https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[9] Parvez. (2016, January 27). *Types of Web Services SOAP,XML-RPC and Restful.* Preluat de pe Php Flow: http://phpflow.com/php/web-service-types-soapxml-rpcrestful/

să facă. Restul cererii sunt doar detalii. Dacă metoda HTTP nu corespunde cu metoda pe care o apelează clientul, atunci serviciul web nu este RESTful. La fel dacă argumentele nu sunt date de URI. Ne vom referi la arhitectura REST în detaliu atunci când vom prezenta principiile REST.

3.4.2 Principii REST

Resurse

Un concept important care ţine de arhitectura REST este existenţa resurselor. În mod uzual o resursă este o entitate care poate fi stocată într-un computer şi poate fi reprezentată printr-un şir de biţi: un document, un rând dintr-o bază de date, sau rezultatul rulării unui algoritm. O resursă poate să fie şi un obiect fizic cum ar fi un măr, un concept abstract cum ar fi curaj, dar reprezentările unor astfel de resurse sunt de cele mai multe ori dezamăgitoare. Fiecare resursă are ataşat un identificator global (un URI). Pentru a manipula resursele, componentele reţelei (clienţii şi serverele) comunică printr-o interfaţă standardizată (de exemplu HTTP) şi schimbă reprezentări ale resurselor (documentele care transportă informaţiile). De exemplu, o resursă care este un cerc ar putea accepta şi returna o reprezentare care specifică un centru şi o rază, formatată în SVG, dar poate accepta şi returna o reprezentare care specifică trei puncte diferite situate pe cerc, separate prin virgulă.

Iată câteva posibile resurse:

- Versiunea 2.0.1 a unui release software
- Ultima versiune a unui release software
- O hartă rutieră a Clujuluii
- Câteva informaţii generale despre leadership
- Următorul număr prim după 1024
- Următoarele zece numere prime după 1024
- Valoarea vânzărilor pentru trimestrul al doi-lea din 2015
- Relaţia dintre două cunoştinţe: Florin şi Răzvan

URI

Orice resursă trebuie să aibă cel puţin un URI. URI-ul este numele şi adresa resursei. Dacă o informaţie nu are un URI, nu este o resursă şi nu putem spune că se află pe Web. URI-urile trebuie să fie descriptive.

Trebuie să existe o corespondență intuitivă între un URI și o resursă. Iată câteva exemple bune de URI-uri pentru resursele introduse mai devreme:

- http://www.xwiki.com/enterprise/releases/2.0.1.tar.gz
- http://www.xwiki.com/enterprise/releases/latest.tar.gz
- http://www.harti.ro/harta/rutiera/Romania/Brasov
- http://en.wikipedia.org/wiki/Leadership

URI-urile trebuie să aibă o structură, adică să varieze într-un mod predictibil. Nu putem merge la caută/leopard pentru resursa leopard şi la vreau-să-ştiu-despre/leu pentru leu. Dacă un client cunoaşte structura URI-urilor unui serviciu, îşi poate crea propriile puncte de intrare în acel serviciu, şi obţine informaţii într-un mod eficient.

După definiţie, două resurse nu pot fi aceleaşi. Dacă acest lucru s-ar întâmpla am avea o singură resursă. Cu toate acestea, la un moment dat, două resurse pot avea aceeaşi reprezentare. Dacă release-ul curent este 2.0.1, atunci http://www.xwiki.com/enterprise/releases/2.0.1.tar.gz şi http://www.xwiki.com/enterprise/releases/latest.tar.gz s-ar referi la acelaşi fişier pentru o perioadă. Dar ideeile din spatele acestor două URI-uri sunt diferite: primul URI este numele unei anumite versiuni, iar al doilea URI este numele unei versiuni care în momentul respectiv este cea mai recentă. Sunt două concepte diferite, deci două resurse diferite.

Adresabilitate

O aplicaţie este adresabilă dacă expune aspecte interesante legate de setul său de date ca şi resurse. Cum resursele sunt expuse prin URI-uri, o aplicaţie adresabilă expune un URI pentru orice informaţie care s-ar putea dovedi de folos. De obicei, se ajunge la un număr foarte mare de URI-uri.

Adresabilitatea este una din cele mai bune caracteristici ale unei aplicaţii web. Permite clienţilor să folosească aplicaţii web în feluri la care dezvoltatorii nu s-au gândit. Urmând acest principiu în dezvoltarea unui serviciu web, utilizatorii serviciului vor beneficia de mai multe avantaje ale REST. Din acest motiv serviciile REST-RPC sunt atât de comune: combină adresabilitatea cu modelul programării bazat pe apeluri de funcţii.

Din păcate, multe aplicaţii web nu respectă principiul adresabilităţii. Acest lucru se întâmplă în special în cazul aplicaţiilor care folosesc AJAX în mod extensiv.

De exemplu, din punctul de vedere al utilizatorilor, există un singur URI pentru Gmail: https://mail.google.com/.

Orice acţiune am face nu vom vedea un alt URI. Resursa “emailurile despre REST” nu este adresabilă, chiar dacă are un URI:

https://mail.google.com/mail/?q=REST&search=query&view=tl.

Problema este că utilizatorul nu este consumatorul sitului web. Situl web este de fapt un serviciu web, iar adevăratul consumator este un program JavaScript care rulează în browserul web. Serviciul web Gmail este adresabil, dar aplicaţie web Gmail care foloseşte acest serviciu nu este adresabilă.

Fără stare

Lipsa stării se traduce prin faptul că orice cerere HTTP se întâmplă într-o izolare completă. Atunci când un client face o cerere HTTP, include informaţiile necesare pentru ca serverul să îndeplinească acea cerere. Serverul nu se bazează niciodată pe cereri precedente.

Mai practic, putem considera lipsa stării în termeni de adresabilitate. Adresabilitatea ne obligă să expunem orice informaţie utilă ca şi resursă cu un URI propriu. Pentru a respecta principiul lipsei stării trebuie să definim ca resurse şi stările posibile pe care le poate avea serverul. Clientul nu trebuie să aducă serverul într-o anumită stare pentru a-l face mai receptiv unei anumite cereri.

Pentru a înţelege mai bine, putem lua ca exemplu un motor de căutare. Când facem o căutare pe Google nu obţinem toate rezultatele ci doar o listă cu primele zece rezultate, care corespund cel mai bine interogării. Pentru a obţine mai multe rezultate trebuie să facem o nouă cerere. Următoarele pagini pe care le obţinem sunt stări diferite ale aplicaţiei, şi au propriul URI: http:// www.google.com/search?q=REST&start=10. Putem transmite această stare a aplicaţiei, putem să îi facem cache, sau o putem adăuga la bookmarkuri, la fel cum putem face cu orice altă resursă adresabilă. Această aplicaţie este fără stare deoarece fiecare cerere este total deconectată de celelelalte. Un client poate cere resursele de mai multe ori, în orice ordine. Poate să ceară pagina doi a listei cu rezultate, înainte să ceară prima pagină, iar pentru server nu va avea nici o importanţă.

Serverul nu trebuie să îşi facă griji că o conexiune ar putea expira, pentru că nici o interacţiune nu dureaază mai mult decât o singură cerere. De asemenea, serverul nu trebuie să cunoască "unde" se află fiecare client în aplicaţie, deoarece clienţii trimit toate informaţiile necesare cu fiecare cerere. Clientul nu va face niciodată o acţiune într-un directoriu nedorit, deoarece serverul a ţinut anumite stări fără să anunţe clientul.

Lipsa stării aduce şi noi funcţionalităţi. Este mai uşor să distribui o aplicaţie fără stare pe mai multe servere. Deoarece două cereri nu depind niciodată una de cealaltă, pot fi tratate de

două servere diferite fără să comunice între ele. Acest aspect imbunătăţeşte scalabilitatea aplicaţiei.

Pentru ca un serviciu să fie adresabil trebuie să disecăm datele aplicaţiei în seturi de resurse. În schimb, HTTP este un protocol fără stare, astfel lipsa stării în serviciul nostru este implicită.

Reprezentări

Atunci când împărţim aplicaţia în resurse, mărim zona de interacţiune cu clienţii. Utilizatorii îşi pot construi un URI potrivit pentru a accesa exact datele de care au nevoie. Dar resursele nu sunt datele, ci doar ideea arhitectului despre cum să împartă datele. Un server web nu poate transmite o idee, trebuie să trimită octeţi, într-un anumit format de fişier, care este reprezentarea unei resurse.

O resursă este o sursă de reprezentări, iar o reprezentare este formată din date despre starea curentă a unei resurse. Multe resurse sunt ele însele date (cum ar fi o listă de buguri deschise), deci o reprezentare evidentă a unor astfel de resurse sunt chiar datele. Serverul ar putea prezenta lista bugurilor deschise printr-un document XML, o pagină web sau text simplu. Valoarea vânzărilor din trimestrul al patru-lea poate fi reprezentat numeric sau printr-un grafic. Pentru orice resursă putem alege între mai multe tipuri de reprezentări.

În cazul unor resurse care sunt obiecte fizice, sau alte lucruri care nu pot fi reduse la informaţii, nu trebuie să ne aşteptăm să avem o fidelitate perfectă a reprezentărilor. Adică reprezentările unor astfel de resurse sunt alcătuite din orice informaţii utile despre starea curentă a resurselor. Dacă considerăm un obiect fizic, cum ar fi un automat de sucuri, având ataşat un serviciu web. Scopul serviciului este de a permite clienţilor automatului să evite drumurile nenecesare până la automat. Folosind acest serviciu, clienţii vor şti când sucul este rece, şi când marca lor preferată de sucuri nu mai este disponibilă. Nimeni nu se aşteaptă ca sucurile să fie disponibilie prin serviciul web, deoarece obiectele fizice nu sunt date. Dar au anumite date ataşate şi anume metadatele. Fiecare slot al automatului poate fi instrumentat astfel încât să cunoască preţul, temperatura şi marca următoarei doze de suc. Fiecare slot poate fi expus ca şi resursă, ca şi tot automatul de altfel. Metadatele pot fi folosite în reprezentările resurselor.

Dacă un server oferă mai multe reprezentări ale unei resurse, atunci se pune problema cum îşi dă seama pe care dintre reprezentări o preferă clientul. De exemplu, un comunicat de presă poate să aibe o versiune în limba engleză şi o versiune în limba spaniolă. Cea mai simplă soluţie este să folosim URI-uri diferite pentru fiecare resursă.

Astfel, http://www.example.com/releases/104.en poate fi reprezentarea în limba engleză a comunicatului, iar http://www.example.com/releases/104.es denotă reprezentarea în limba

spaniolă. Dezavantajul este faptul că expunem URI-uri diferite pentru aceeaşi resursă, iar oamenii care vorbesc despre comunicatul de presă în limbi diferite par ca vorbesc despre lucruri diferite. Alternativa constă în negocierea conţinutului. În acest scenariu avem un singur URI expus şi anume http://www.example.com/releases/104. Când un client face o cerere către acest URI, oferă antete HTTP speciale care specifică ce fel de reprezentări acceptă.

Interfaţa uniformă

În spaţiul World Wide Web, sunt doar câteva acţiuni pe care le poţi face cu o resursă. HTTP oferă patru metode de bază pentru cele mai comune operaţii:

- Returnarea unei reprezentări a unei resurse: HTTP GET
- Crearea unei noi resurse: HTTP PUT unui nou URI, or HTTP POST unui URI existent
- Modificarea unei resurse: HTTP PUT unui URI existent
- Ştergerea unei resurse: HTTP DELETE

Pentru a obţine sau a şterge o resursă, clientul trimite o cerere GET sau DELETE asupra URI+ului resursei. În cazul unei cereri GET, serverul returnează o reprezentare în corpul răspunsului. Pentru o cerere DELETE, corpul răspunsului poate conţine un mesaj de stare, sau nimic.

Pentru a crea sau modifica resurse, clientul trimite o cerere PUT care, de obicei, include o entitate-corp. Entitatea-corp conţine propunerea clientului pentru reprezentarea resursei. Ce fel de date alcătuiesc reprezentarea, şi în ce format, depinde de serviciu.

Într-o arhitectură REST, POST este folosit pentru a crea resurse subordonate: resurse sunt în relaţie cu o altă resursă "părinte". De exemplu, pentru a crea o pagină wiki într-un spaţiu XWiki, trimitem o cerere POST resursei părinte, care este spaţiul.

Mai avem două metode HTTP care pot fi folosite într-un serviciu web:

- Returnarea unei reprezentări formată doar din metadate: HTTP HEAD
- Verificarea metodelor HTTP suportate de o anumită resursă: HTTP OPTIONS

Pentru ca un serviciu web să fie RESTful nu trebuie să folosim interfaţa uniformă definită de HTTP. Metodele HTTP nu sunt cele mai potrivite de fiecare dată. Ceea ce contează este să alegem o interfaţă care să fie uniformă. Dacă avem un URI al unei resurse, ştim cum obţinem reprezentarea: trimitem o cerere HTTP GET acelui URI. Interfaţa uniformă face ca două servicii să fie la fel de similare cum sunt două situri web. Fără interfaţa uniformă, ar trebui să aflăm cum fiecare serviciu web acceptă şi transmite informaţii. Regulile ar putea fi diferite chiar şi pentru diferite resurse dintr-un singur serviciu. Fără o interfaţă uniformă, am avea numeroase metode care ar lua locul lui GET: doSearch, getPage, nextPrime.

Câteva aplicaţii extind interfaţa uniformă HTTP. Cea mai cunoscută este WebDav, care adaugă opt noi metode HTTP, printre care MOVE, COPY şi SEARCH. Dacă am folosi aceste metode într-un serviciu web nu am viola nici un principiu REST, pentru că arhitectura REST nu impune folosirea unei anumite interfeţe uniforme.

Siguranţa şi idempotenţa

Dacă în dezvoltarea unui serviciu web folosim ca interfaţă uniformă HTTP, obţinem două proprietăţi folositoare. Când sunt folosite corect, cererile GET şi HEAD sunt sigure. Iar cererile GET, HEAD, PUT şi DELETE sunt idempotente.

O cerere GET sau HEAD se face pentru a citi anumite date, nu pentru a schimba ceva pe server. Clientul poate să facă o cerere GET sau HEAD de zece ori, iar resursa se comportă de parcă nu s-ar fi făcut nici o cerere. Faptul că o cerere este sigură se traduce prin faptul că doar se returnează o reprezentare a resursei. Un client ar trebui să aibe posibilitatea de a trimite cereri GET şi HEAD asupra unui URI necunoscut, fără să se întâmple ceva nedorit.

Idempotenţa este o idee un pic mai greu de perceput decât siguranţa. Ideea vine din matematică, unde o operaţie este idempotentă dacă are acelaşi efect chiar dacă o aplici de mai multe ori. Multiplicarea unui număr cu zero este idempotentă: 4 ×0 ×0 ×0 este acela;i lucru cu 4 ×0. Prin analogie, o operaţie asupra unei resurse este idempotentă dacă a face o singură cerere este similar cu a face o serie de cereri identice. A doua cerere şi următoarele lasă resursa în aceeaşi stare pe care o avea după prima cerere.

PUT şi DELETE sunt idempotente. Dacă ştergem o resursă, aceasta dispare. Dacă facem o nouă cerere DELETE, ea rămâne ştearsă. La fel, dacă creăm o nouă resursă cu PUT, şi apoi retrimitem cererea PUT, resursa va rămâne la fel şi cu aceleaşi proprietăţi ca după creare. Dacă folosim PUT pentru a schimba starea unei resurse, putem retrimite cererea PUT iar starea resursei va rămâne aceeaşi.

Partea practică a idempotenţei este faptul că nu ar trebui să lăsăm clienţii să schimbe starea unei resurse în termeni relativi. Dacă o resursă reţine o valoare numerică ca parte din starea resursei, un client ar putea folosi PUT pentru a seta această valoare pe 4, sau 0, sau alt număr, dar nu pentru a incrementa valoarea. Dacă valoarea iniţială este 0, dacă trimitem două cereri PUT care să seteze valoarea la 4, valoarea numerică rămâne 4. Dar dacă trimitem două cereri PUT prin care incrementăm valoarea numerică cu 1, atunci valoarea nu va rămâne 1, ci va fi 2, ceea ce nu respectă idempotenţa.

Siguranţa şi idempotenţa oferă posibilitatea unui client să facă cereri HTTP sigure pe o reţea nesigură. Dacă facem o cerere GET şi nu primim nici un răspuns, putem să facem o nouă cerere fără nici o problemă. La fel se întâmplă si cu o cerere PUT. Metoda POST, în schimb, nu

este nici sigură şi nici idempotentă. Dacă facem două cereri POST asupra unei resurse, cel mai probabil, vom obţine două resurse subordonate, conţinând aceeaşi informaţie.

O greşeală comună într-un serviciu web care se doreşte să respecte principiile REST, este de a expune operaţii nesigure prin metoda GET. API-urile Del.icio.us şi Flickr fac această greşeală. Atunci când facem o cerere GET pe https://api.del.icio.us/posts/delete, nu obţinem o reprezentare, ci modificăm datele de pe server.

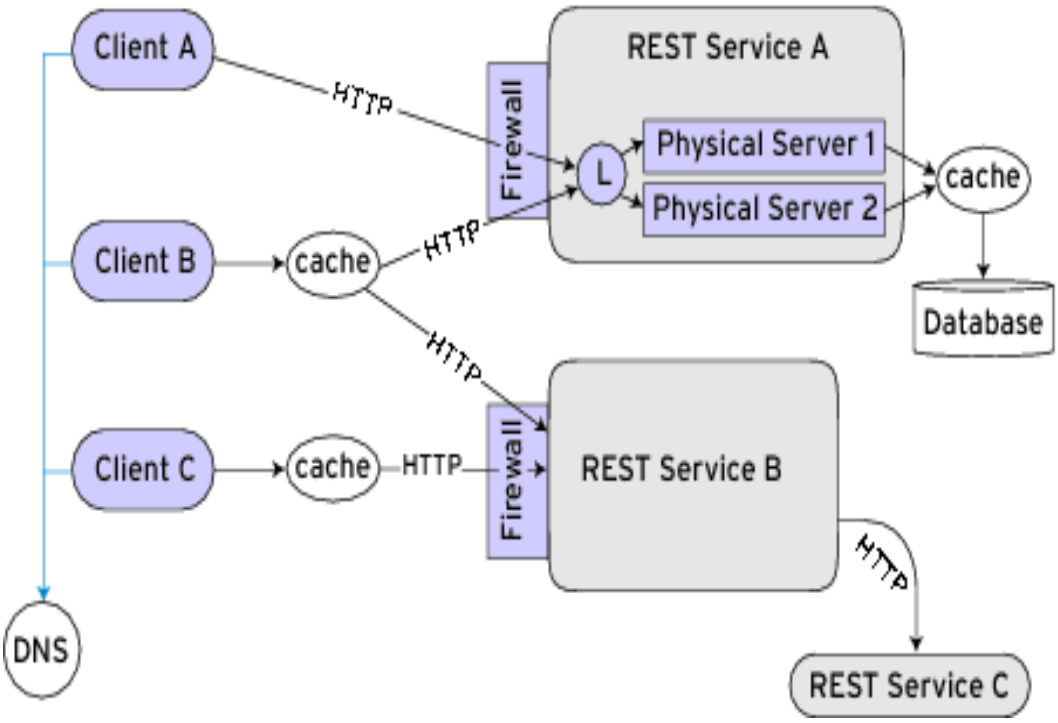


Fig. 6 Diagrama arhitecturii REST[10]

3.4.3 Concluzii și avantaje

Beneficiile utilizării arhtecturii REST în dezvoltarea serviciilor web sunt multiple. Datorită faptului că reprezentările pot fi cache-uite timpul de răspuns al serverului şi încărcarea acestuia sunt reduse. Scalabilitatea serverului este îmbunătăţită reducându-se nevoia ca serverul să menţină anumite stări care ţin de o sesiune. Astfel servere diferite pot fi folosite pentru a trata diferite cereri dintr-o sesiune.

[10] Bruno, E. J. (2007, June 08). *SOA, Web Services, and RESTful Systems.* Preluat de pe Drdobbs: http://www.drdobbs.com/web-development/soa-web-services-and-restful-systems/199902676

Codul de pe client este redus deoarece browserul poate fi folosit pentru a accesa orice resursă. De asemenea, un serviciu web RESTful depinde mai puţin de formate proprietare şi de frameworkuri de mesagerie deasupra HTTP.

Aplicațiile REST sunt simple, *lightweight* și rapide pentru că[11]:

- Resursele sunt identificate prin URI, ceea ce furnizează o modalitate de adresare globală.
- O interfață uniformă este folosită pentru manipularea resursei.
- Sunt folosite mesaje autodescriptive sau metadate pentru resurse.
- Interacțiunile *stateful* prin *hiperlinkuri* sunt bazate pe conceptul de stare explicită de transfer.

Cateva exemple de implementari de servicii REST sunt prezentate in lista de mai jos:

- Amazon's Simple Storage Service (S3) (*http://aws.amazon.com/s3*)
- Servicii care expun protocolul Atom Publishing Protocol (*http://www.ietf.org/html.charters/atompub-charter.html*)
- Majoritatea serviciilor web de la Yahoo (*http://developer.yahoo.com/*)

[11] Silviu Dumitrescu, D. B. (n.d.). JavaFX și comunicarea prin RESTful Web Services. *Today Software Magazine*.

4. Tehnologii folosite

4.1 Framework-ul open-source JavaScript AngularJS

4.1.1 Prezentare generală

AngularJs este un freamework JavasScript „cu super puteri” aşa cum se autointitulează, dezvoltat de compania Google cu scopul de a imbunatății semnificativ experiența utilizatorului în cadrul aplicațiilor web. Principiile şi standardele tradiționale de succes ce stau la baza proiectării și dezvoltării aplicațiilor server se regasesc si în modelul arhitectural propus de AngularJs fapt pentru care procesul de dezvoltare al aplicațiilor client este mult mai rapid. Accest aspect conferă un grad mare de scalabiliate ce asigură o proiectare robustă a aplicațiilor cu un nivel mare de complexe.

Filozofiea AngularJs constă în parcurgerea unor puncte cheie pornind de la stucturarea aplicației la cum ar trebui aceasta să fie construită şi legată, la testarea aplicației şi integrarea codului cu alte librării.[12] Din punct de vedere structural, infrastructura aplicației AngularJS este descrisă de sabloanele de proiectare MVC(Model-View-Controller) şi MVVM (Model-View-ViewModel).

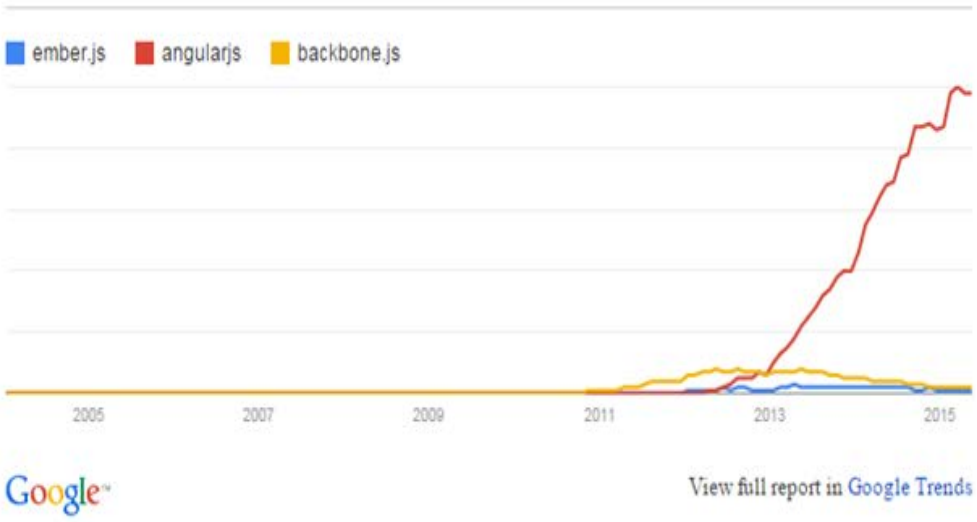


Fig. 7 Rata de creștere și dezvoltare a framework-urilor, evidențiată folosind Google Trends

[12] Shyam Seshadri, B. G. (2014). *AngularJS: Up And Running.* September: O'Reilly Media.

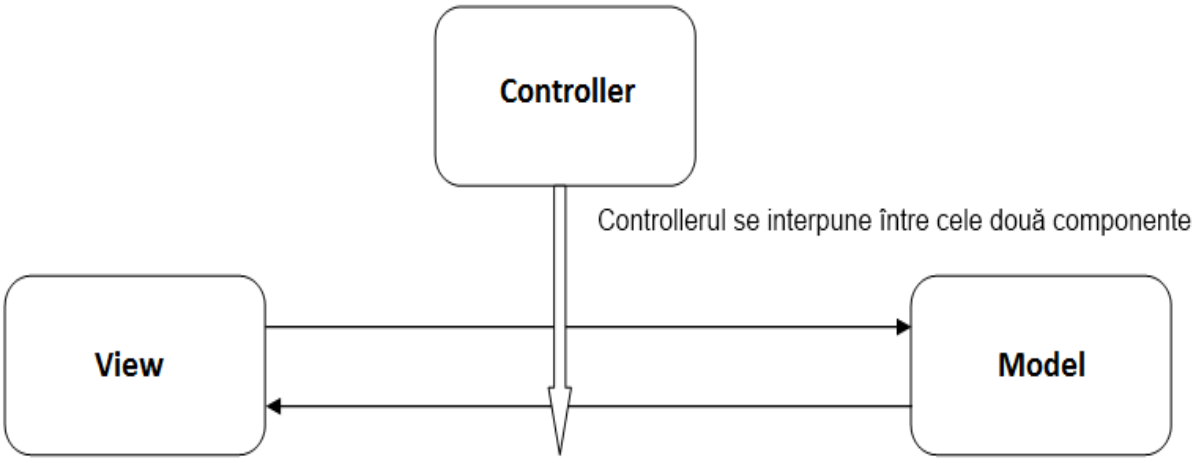
4.1.2 Șablonul arhitectural Model-View-Controller

MVC, sau Model-View-Controller este un șablon arhitectural folosit în proiectarea si dezvoltarea aplicațiilor. Aceasta modalitate de lucru reușeste cu succes izolarea parții logice de interfața proiectului, rezultând in aplicații extrem de usor de modificat. În organizarea MVC, modelul reprezinta informația (datele) de care are nevoie aplicația, viewerul corespunde cu elementele de interfață iar controller-ul reprezintă sistemul comunicativ si decizional ce procesează datele informaționale, facand legatura între model și view.

Modelul reprezintă partea de "hard-programming", partea logică a aplicației. El are în responsabilitate acțiunile si operațiile asupra datelor, autentificarea utilizatorilor, integrarea diverselor clase ce permit procesarea informațiilor din diverse baze de date.

View-ul se ocupă de afisarea datelor, practic aceasta parte a programului va avea grijă de cum vede end-userul informația procesată de controller. Odată ce funcțiile sunt executate de model, viewului îi sunt oferite rezultatele, iar acesta le va trimite către browser. În general viewul este o mini-aplicație ce ajută la randarea unor informații, având la bază diverse template-uri.

Controller-ul reprezintă creierul aplicației. Aceasta face legătura intre model si view, între acțiunile userului și partea decizională a aplicației. In funcție de nevoile utilizatorului, controllerul apelează diverse funcții definite special pentru secțiunea de site în care se află userul. Funcția se va folosi de model pentru a prelucra (extrage, actualiza) datele, după care informațiile noi vor fi trimise catre view, ce le va afișa apoi prin template-uri.



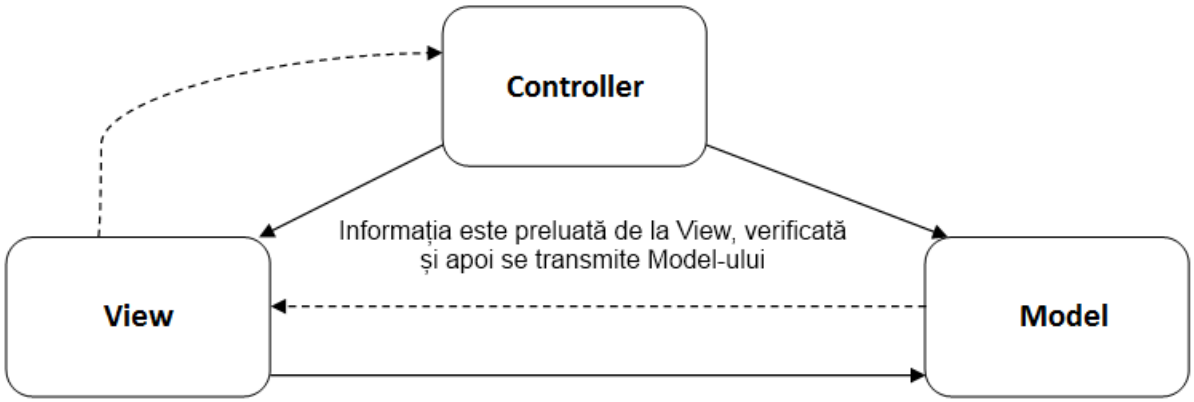


Fig. 8 Arhitectura MVC

4.1.3 Șablonul arhitectural Model-View-ViewModel

Model-View-ViewModel (MVVM) este model arhitectural derivat din MVC. La fel ca și în cazul modelului Pasive-View nu există o dependență între view și model, dar spre deosebire de acesta view-ul nu este pasiv ci poate actualiza controller-ul care în acest caz este reprezentat de ViewModel. Cu toate acestea din punct de vedere al testării automate doar ViewModel-ul trebuie testat, întrucât comunicarea între View și ViewModel se face prin data binding, care cel puțin teoretic nu poate conține erori de logică. Modelul arhitectural Model-View-ViewModel este un model specific platformelor .Net: Windows Presentation Foundation și Silverlight. Pentru a ușura dezvoltarea aplicațiilor MVVM au fost dezvoltate o serie de toolkit-uri care facilitează implementarea acestui model arhitectural. Unul din cele mai cunoscute astfel de toolkit-uri este MVVM Light Toolkit.[13]

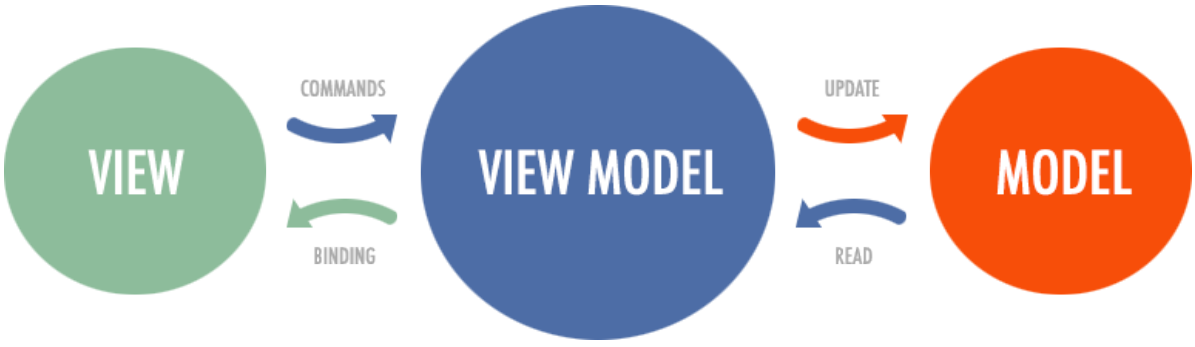


Fig. 9 Arhitectura MVVM[14]

[13] http://mvvmlight.codeplex.com/
[14] https://erazerbrecht.wordpress.com/2015/10/13/mvvm-entityframework/

4.1.4 Concluzii și avantaje

Unul dintre cele mai mari avantaje AngularJS îl constituie echipa de dezvoltare și întreținere a platformei, care îi conferă beneficii precum performanța, robustețea și scalabilitatea codului și îl face ca platforma să fie de integrat cu proiectele web.

Un alt rezultat bun constă în actualizarea regulată a platformei cu ultimele noutăți în domeniul dezvoltării web. Deși AngularJS a fost preluat de Google în 2010, platforma este încă open-source ceea ce reprezintă tot un avantaj, întrucât ea ramâne gratuită. Structura aplicațiilor dezvoltate cu Angular este intuitivă și framework-ul este suficient de flexibil pentru a ușura și a crește viteza de dezvoltare a proiectelor.

Platforma impune un mod de dezvoltare după o structură MVVM (Model View View Model), design pattern care preia modularitatea din conceptele MVC și face extrem de ușoară integrarea template-urilor HTML cu procesările ce se fac pe Javascript (prin two-way data-binding).

Modulele create la nivel de Javascript în Angular sunt ușor de integrat deoarece framework-ul vine cu capacitatea de injectare a dependințelor, ceea ce contribuie și la scalabilitatea codului. Viteza de incărcare a paginilor este mică întrucât majoritatea comunicației cu serverul este asincronă iar apelurile la server sunt reduse, majoritatea interfeței vizuale fiind mutată pe client (în browser). AngularJS permite crearea de aplicații responsive și îmbunătățeste experiența utilizatorului pe site datorită platformei sale flexibile, poate fi ușor integrată cu alte framework-uri de testare automată

Rezultatul final : o aplicație fluidă, suplă, dezvoltată în timp scurt, care regrupează o mare varietate de efecte vizuale și animații grafice excepționale. Este ușor de intreținut și beneficiază de o viteză mare de incărcare. AngularJS este performant, intuitiv, atractiv și corespunde perfect platformelor mobile.

4.2 Microsoft ASP.NET

4.2.1 Introducere

ASP.NET este un set de tehnologii care ne permit crearea de aplicaţii web. Este evoluţia de la Microsoft Active Server Pages (ASP), dar beneficiază de suportul platformei de dezvoltare Microsoft .NET.

Una dintre cele mai importante calităţi ale ASP.NET este timpul redus necesar dezvoltării aplicaţiilor web. Atât tehnologia în sine, cât şi uneltele de dezvoltare de aplicaţii web de la Microsoft reduc considerabil timpul de dezvoltare al aplicaţiilor web faţă de alte tehnologii – prin simplitatea unui limbaj de programare ”managed” de genul C# sau Visual Basic .NET, prin colecţia bogată de biblioteci de clase şi controale .NET care oferă foarte multă funcţionalitate ”out of the box”, prin orientarea pe construirea de aplicaţii web a mediului de dezvoltare VWD.

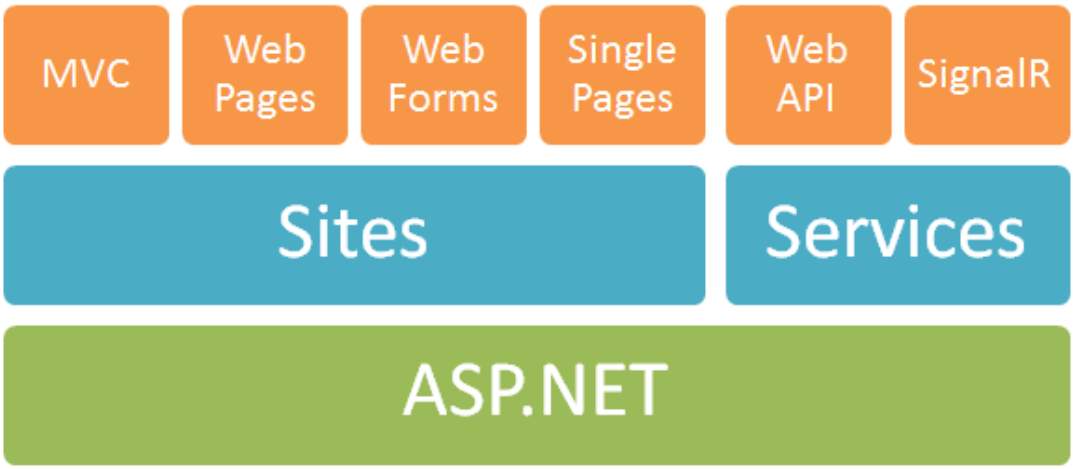


Fig.10 Colectia de framework-uri ASP.NET [15]

4.2.2 Noțiunea de interfață de programare (API)

Termenul de API este acronimul de la ***A**pplication**P**rograming **I**nterface* care în limba romană se poate traduce ca interfață de programare a aplicațiilor şi reprezintă un set de reguli şi specificații cu ajutorul carora un program poate accesa și folosi resursele oferite de un alt program. Cu alte cuvinte, un program care oferă API-uri permite altor programe sa interacționeze cu acesta automat, de la program la program sau de la program la sistem de operare, fară să fie nevoie de folosirea unei interfețe grafice de utilizare (GUI) si fara cunoștințe

15 http://www.hanselman.com/blog/OneASPNETMakingJSONWebAPIsWithASPNETMVC4BetaAndASPNETWebAPI.aspx

legate de arhitectura sau elemente de programare ale programului ci doar folosirea specificațiilor de utilizare a API-ului oferit.

Sunt mai multe tipuri de API-uri în funție de sistemul care le implementează. Exista API-uri pentru sisteme de operare, aplicatii software sau aplicații web.

De exemplu, sistemul de operare Microsoft Windows ofera API-uri pentru:

- programele ce operează asupra dispozitivelor instalate în sistem (drivere) pentru a putea folosi funcțiile acestor dispozitive;
- programele care rulează - un exemplu de astfel de interacțiune este atunci când efectuati copy - paste asupra unui text dintr-un program într-altul: atunci când efectuați copiere programul va stoca informația in Windows prin intermediul unui API cu transferul de date de la program la sistemul de operare, cand efectuati lipire programul va apela un API din Windwos pentru a transfera textul copiat din sistemul de operare in programul unde s-a initiat aceasta actiune;
- un mediu de operare prin care programatori pot folosi API-uri (comenzi MS-DOS) pentru crearea unei aplicatii care sa ruleze in mediul in care a fost programat.

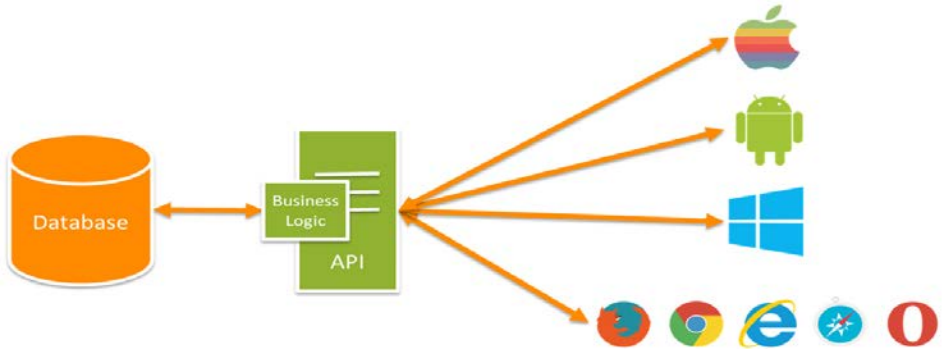


Fig. 11 Privire de ansamblu asupra functionalitătii API-urilor[16]

Acelasi principiu de functionare se poate aplica nu doar intre un program si sistemul de operare, dar si direct intre programe. In plus, lucrul cu API-uri este foarte folosit si in mediul online, o multime de aplicatii web furnizeaza accesul la serviciile oferite prin intermediul unor seturi de reguli de comunicare bine precizate. Exemple de API-uri folosite in mediul online:

- Google Maps API - permite dezvoltatorilor web sa incorporeze harti in paginile web folosid cod JavaScript sau Flash;

[16] Kearn, M. (2015, January 5). *Introduction to REST and .net Web API*. Retrieved from https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/

- YouTube API - permite incorporarea de videoclipuri de pe YouTube pe site-uri externe acestuia;
- Flickr API - folosirea pozelor distribuite de o comunitate Flickr;
- Twiter API, Amazon API, Facebook API si multe altele;

4.2.3 Microsoft ASP.NET Web API

Microsoft ASP.NET Web API un framework server-side open souurce care face parte din suita ASP.NET, folosit pentru a dezvolta servicii web bazate pe modelul arhitectural REST. Intuitiv un serviciu web construit folosind componenta ASP.NET Web API va fi gazduit de un server IIS(Internet Information Service) si va respecta ciclul de viata descris de versiunea aferentă serverului configurat.

Prima versiune a fost lansată în august 2012 si a venit impreună cu Visual Studio 2012, .Net 4.5 si ASP.NET MVC. Datorită interacțiunii usoare cu protocolul HTTP proiectarea unui serviciu web devine foarte accesibilă, oferind un instrument solid pentru implementarea unei arhitecturi de tip REST.

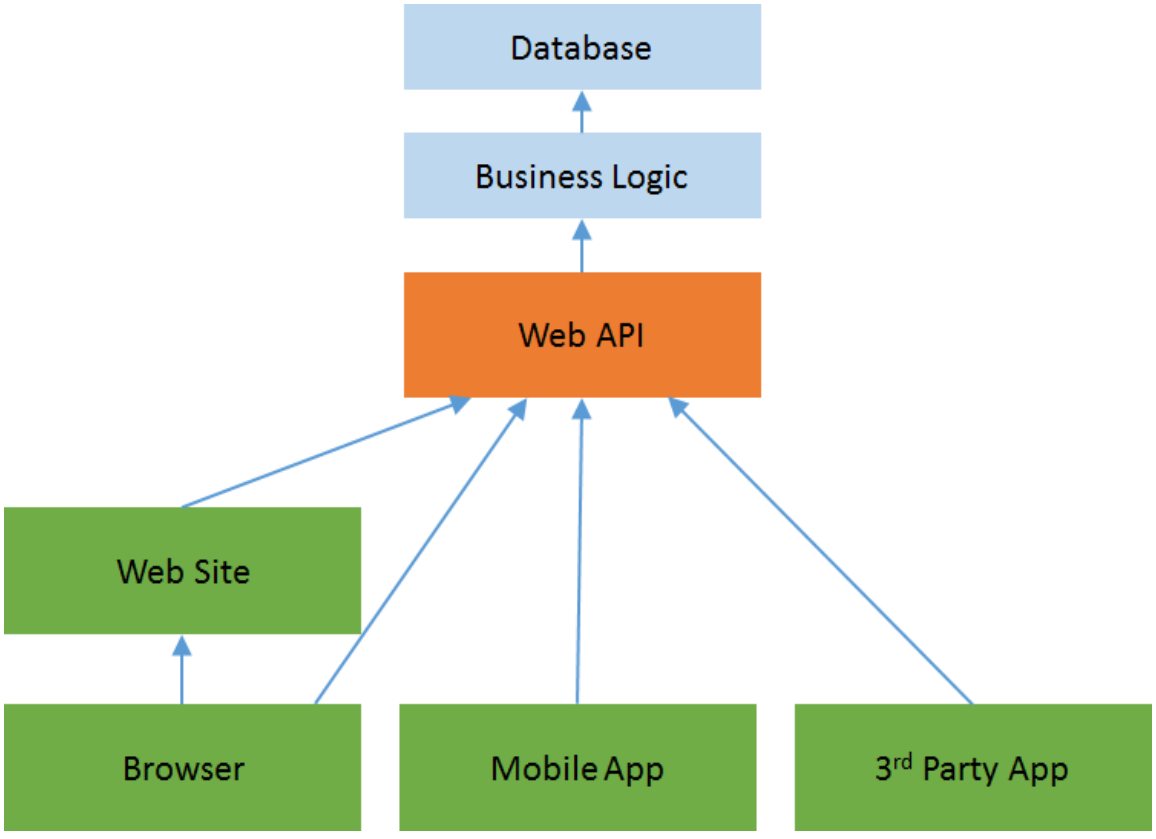


Fig. 12 Interoperabilitatea unui API

Pe scurt, pentru a răspunde unei cereri HTTP de tipul GET adresată de URL-ul http://server/hrapp/employees/12345 este nevoie de a defini o clasă scrisă intr-un limbaj acceptat de tehnologia ASP.NET Web API, C# ori VisualBasic ce va fi derivată din clasa ApiController si care va implementa o metoda GET. Metoda definită trebuie sa conțină în antet un parametru id care indica id-ul angajatului. Datorită conccptului de rutare cu care vine acest frameork in functie de prototipul functiei si tipul metodei descrise de cerere se va executa cu succes metoda țintită. In corpul funcției se va implementa logica de returnare a obiectului care va fi serializat la preferința programatorului in format JSON ori XML, in functie de atributul ce negociază raspunsul definit in cererea inițiată (content-type).

ASP.NET Web Api are la baza multe dintre conceptele si particularitațile de design folosite de framework-ul ASP.NET MVC, cum sunt controllere-le, acțiunile, filtrele, bindere-le, etc.

4.3 Microsoft SQL Server

Microsoft Sql Server, este un sistem din clasa Enterprise ce se poate aplica bazelor de date de dimensiuni foarte mari. SQL Server este un DBMS (Data Base Management System) – sistem pentru gestiunea bazelor de date, produs de Microsoft. Suporta versiunea Microsoft de SQL (Structured Query Language) - limbaj structurat de interogări, cel mai comun limbaj pentru bazele de date.

Codul de baza pentru Microsoft SQL Server deriva din Sybase SQL Server si a reprezentat intrarea Microsoft pe piaţa bazelor de date la nivel enterprise, concurând cu Oracle, IBM si Sybase. Microsoft, Sybase si Ashton-Tate s-au unificat pentru a crea si a scoate pe piaţa prima versiune numita SQL Server 4.2 ptr Win OS/2. Mai tarziu Microsoft a negociat pentru drepturi de exclusivitate la toate versiunile de SQL Server scrise pentru sistemele de operare Microsoft. Sybase si-a schimbat ulterior numele in Adaptive Server Enterprise pentru a se evita confuzia cu Microsoft SQL Server.SQL Server 7.0 a fost primul server de baze de date bazat pe GUI. O varianta de SQL Server 2000 a fost prima varianta comerciala pentru arhitectura Intel. Ultima versiune aparuta este Microsoft SQL Server 2008.

Microsoft SQL Sever foloseşte o varianta de SQL numita T-SQL, sau Transact-SQL, o implementare de SQL-92 (standardul ISO pentru SQL) cu unele extensii. T-SQL in principal adăuga sintaxa adiţionala pentru procedurile stocate si pentru tranzacţii. Standardele SQL necesita ACID - patru condiţii pentru orice tranzacţie, sintetizate prin acronimul ACID: atomicitate, consistenta, izolare, durabilitate.

Microsoft aduce o serie de îmbunătăţiri precum support pentru gestionarea de date XML, in plus faţă de date relaţionale. Metode de indexare specializate au fost puse la dispoziţia datelor XML, iar interogarea lor se face folosind XQuery. Sql Server 2005 adăuga unele extensii

limbajului T-SQL precum funcţii de eroare a manipulării si suport pentru interogările recursive. Permisiunile si controlul accesului au mai

Versiunea SQL Server 2008 lansata pe 6 august 2008 cu nume de cod Katmai oferă suport pentru stocarea datelor multimedia si adăuga noi tipuri de date (geometry, geography, hierarchy si mult asteptatul date fara datetime). Versiunea de SQL Server Management Studio inclusa in SQL Server 2008 acceptă IntelliSense pentru SQL.

Microsoft oferă SQL Server Expess Edition este o versiune gratuita a serverului si nu oferă restricţii in ceea ce priveşte numărul bazelor de date sau a utilizatorilor concurenţi, este limitata la folosirea unui singur procesor, a 1 Gb de memorie si max. 4Gb a fişierelor de date.

Microsoft SQL Server ca este o soluţie integrată de management şi analiză a datelor, care ajuta organizaţiile de orice dimensiune să:

- Dezvolte, implementeze şi administreze aplicaţii la nivel de întreprindere mai sigure, scalabile şi fiabile
- Maximizeze productivitatea IT prin reducerea complexităţii creării, implementării şi administrării aplicaţiilor pentru baze de date.
- Partajeze date pe mai multe platforme, aplicaţii şi dispozitive pentru a facilita conectarea sistemelor interne şi externe.
- Controleze costurile fără a sacrifica performanţa, disponibilitatea, scalabilitatea sau securitatea.

Gestionarea serverului se face foarte uşor prin aplicaţia SQL Server Management Studio Elementul central al acestei unelte este panelul Object Explorer, ce permite utilizatorului sa răsfoiască, selecteze sau sa întreprindă orice alta acţiune asupra obiectelor de pe server.

5. Dezvoltarea aplicativă

5.1 Accesarea resurselor pe baza URL si a protocolului HTTP

Aplicația My Personal Development Plan se bazează pe o arhitectură REST, având în centru un serviciu care acopera urmatoarele functionalitați:

- Gestiunea utilizatorilor - operații CRUD asupra profilului utilizatorului
- Definirea, monitorizarea si gestiunarea scopurilor – operaţtii CRUD
- Definirea, monitorizarea si gestiunarea obiectivelor – operații CRUD
- Gestiunea efortului de timp asupra obiectivelor
- Analiza statistică a scopurilor și a obiectivelor
- Gestiunea unor note informative - operații CRUD
- Gestiunea unui planificator de evenimente – operații CRUD

Implementarea serviciului este realizată folosind framework-ul ASP.NET Web API, un instrument foarte eficient, acesta fiind proiectat exclusiv pentru a dezvolta servicii ce au la baza arhitectura REST. Simplitatea accesării resurselor prin Uniform Resource Locator optimizează timpul de dezvoltare a aplicației, totodată multitudinea de componente cu care vine acest framework asigura posibilitatea unei proiectari complexe si performante. Resursele din cadrul aplicației MPDP sunt serializate in format JSON.

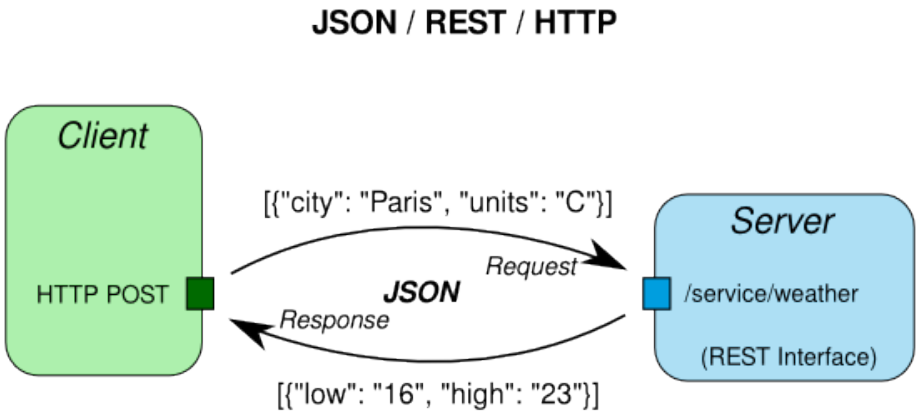


Fig. 13 Accesarea unei resurse in fomat JSON prin HTTP in cadrul arhitecturii REST

5.2 Structura URLului

URL-ul de baza al serviciului web este format dintr-un IP şi un port. În cazul staţiei pe care rulează serverul, acest URL are forma http://localhost:43504/ numele “localhost” fiind translatat local in ip-ul 127.0.0.1. În cazul clienţilor, aceştia vor avea posibilitatea accesării serviciului prin ip-ul public al acestuia, care se setează indepenent de server la rularea acestuia, în funcţie de setările ruterului wireless care asigură comunicaţia.

Pentru aplicaţia My Prsonal Development Plan, am definit mai mult tipuri de URL-uri, în funcţie de resursele accesate. Acestea sunt prezentate în continuare prin porţiunea caracteristică resursei, care se adăugă adresei de bază:

- api/account/login - accesarea credentialelor unui utilizator
- api/goal/getgoals - preluarea tuturor scopurilor a unui utilizator
- api/userprofile/get - accesarea unui profil de utilizator

Pe lângă cele de mai sus, am mai definit patru resurse care sunt create şi actualizate de server:

- api/account/register - inregistrarea unui nou utilizator
- api/goal/creategoal - crearea unui scop
- api/goal/updategoals - modificarea unui scop
- api/goal/createobjective - crearea unui obiectiv si asignarea lui in cadrul unui scop
- api/goal/updateobjective - modificarea unui obiectiv
- api/goal/addworkedlog - adaugarea efortului de timp in cadrul unui obiectiv

5.3 Tipurile mesajelor HTTP

Pentru transmiterea diferitelor mesaje HTTP între client şi serviciu, am folosit setarea antetului Content-Type al mesajului http. In cadrum Aplicației My Personal Development Plan datele sunt serializate in format JSON.

JSON este un acronim în limba engleză pentru *JavaScript Object Notation*, și este un format de reprezentare și interschimb de date între aplicații informatice. Este un format text, inteligibil pentru oameni, utilizat pentru reprezentarea obiectelor și a altor structuri de date și este folosit în special pentru a transmite date structurate prin rețea, procesul purtând numele de serializare. JSON este alternativa mai simplă, mai facilă decât limbajul XML. Eleganța formatului JSON provine din faptul că este un subset al limbajului JavaScript(ECMA-262 3rd Edition) , fiind utilizat alături de acest limbaj. Formatul JSON a fost creat de Douglas

Crockford și standardizat prin RFC 4627. Tipul de media pe care trebuie să îl transmită un document JSON este *application/json*. Extensia fișierelor JSON este *.json*.

Avantaje folosind serializarea JSON:

- Ofera o modalitate automata de a serializa / deserializa obiecte JavaScript, folosind linii minime de cod. În comparație, developerii trebuie să scrie cod JavaScript pentru serializare în XML.
- Este sustinut de toate browserele, parsarea intre browsere la XML se poate dovedi dificila
- Oferă un format concis, datorita abordarii bazate pe perechea nume-valoare
- Deserializare rapidă a obiectelor în JavaScript
- Sustinut de multe toolkit-uri Ajax si biblioteci JavaScript
- API-uri simple, disponibile pentru JavaScript si alte limbaje de programare

5.4 Implementarea protocolului de comunicaţie

Un protocol de comunicaţie se defineşte ca un set de reguli cunoscute de ambele părţi ale unei comunicaţii, prin care atât receptorul, cât şi emiţătorul pot interpreta corect mesajele transmise pentru îndeplinirea anumitor cerinţe. Un protocol defineşte de asemenea şi ordinea mesajelor schimbate, anumite operaţii necesitând schimbul unui număr consecutiv de mesaje specifice, într-o ordine particulară. Un protocol bazat pe HTTP, aşa cum este cazul la aplicaţia My Personal Development Plan, este format din două componente:

1. URL – atât clienţii care accesează resursa cât şi serverul pe care este stocată aceasta au aceeaşi concepţie asupra resursei aflate la acea adresa. Astfel, clientul știe ce adresa trebuie accesată pentru obţinerea resursei dorite şi are certitudinea corectitudinii mesajului primit ca răspuns. Această certitudine permite interpretarea corectă a răspunsului. La fel, serverul asigură stocarea resursei cerute la respectiva adresă, având acelaşi set de mapări resursă-URL ca cel cunoscut de client.

2. Conţinutul mesajelor – în cadrul unei resurse aflate la un URL, se încapsulează un conţinut propriuzis al mesajului care conţine informaţia dorită. Această informaţie poate reprezenta date obţinute prin prelucrare pe server sau pur şi simplu stocate pe acesta. În funcţie de conţinutul acestor mesaje, un client poate decide transmiterea unor noi cereri sau declanşarea unor evenimente locale.

5.5 Arhitectura aplicației

5.5.1 Prezentare

Așa cum am mai precizat aplicația My Personal Development Plan foloseste un serviciu central dezvoltat pe o arhitectura REST implementat folosind tehnologia Microsoft ASP.NET Web Api. La nivel de interacțiune, serviciul implementat în aplicația propusă va fi consumat de catre un client implementat folosind framework-ul Angular, printr-un browser web. Pentru

accesarea şi modificarea resurselor se folosesc mesaje HTTP cu metodele descrise într-un capitol anterior.

Arhitecura aplicației din perspectiva tehnologiilor implementate, a logicii de funcționare şi a acesării datelor este descrisă de figura de mai jos:

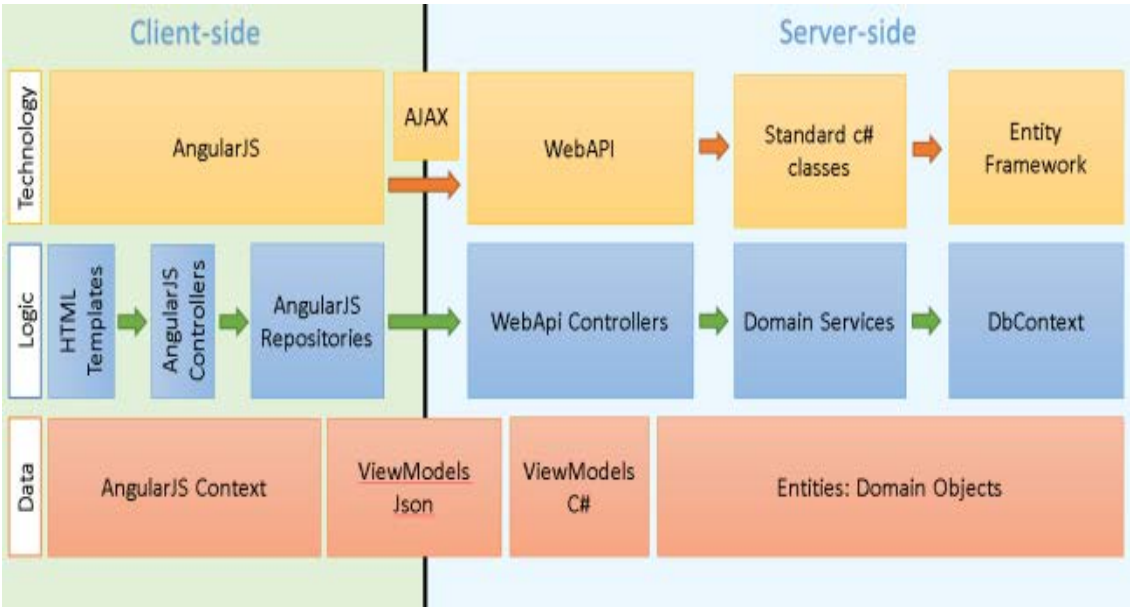


Fig. 14 Diagrama de interacțiune prezentată din mai multe perspective

La nivel structural în cadrul modelului client/server, aplicația este distribuită pe trei straturi în trasabilitate. Modelul arhitectural descris pe trei nivele implică fragmentarea aplicației pe trei module diferite.

Primul nivel (clientul) al arhitecturii client/server pe trei niveluri este reprezentat de interfața sistemului cu utilizatorul (sesiuni de lucru, ferestre de dialog, ferestre pentru introducerea datelor, administrarea afişării etc.), adică logica prezentării. In cazul aplicației My Personal Development Plan acest nivel este ocupat de aplicația dezvoltata cu tehnologia Agular.

Nivelul de mijloc furnizează servicii de administrare a regulilor de business şi a prelucrării datelor care pot fi partajate de mai multe aplicaţii. Acest nivel de mijloc asigură performanţă, flexibilitate, mentenabilitate, reutilizabilitate şi scalabilitate prin centralizarea logică a proceselor. Această centralizare face ca administrarea şi gestiunea modificărilor să fie mai uşoară prin localizarea funcţionalităţii sistemului astfel încât modificările sunt executate o singură dată şi plasate pe server-ul nivelului de mijloc pentru a fi disponibile de-a lungul întregului sistem. În plus, acest nivel controlează tranzacţiile şi interogările asincrone pentru a se asigura că acestea sunt efectuate complet.

Al treilea nivel (server-ul de baze de date) furnizează funcţionalitatea gestiunii bazei de date şi este dedicată serviciilor de fişiere şi date, respectiv logica accesului la baza de date şi validarea din partea server-ului. Componenta de gestiune a datelor garantează că datele sunt consistente de-a lungul mediului distribuit prin utilizarea unor caracteristici precum blocarea datelor, consistenţa şi replicarea acestora. Conectivitatea între niveluri poate fi modificată în mod dinamic în funcţie de cererile utilizatorilor pentru date şi servicii.

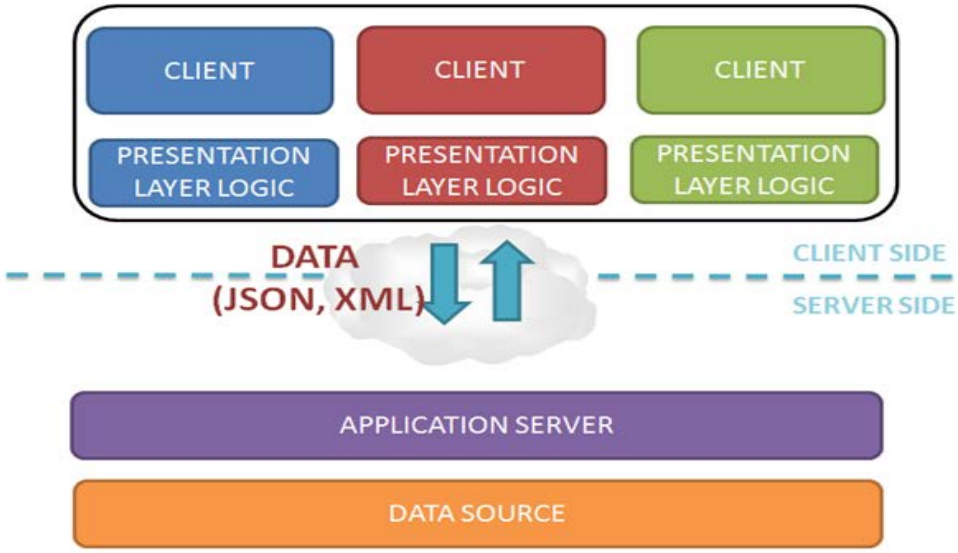


Fig. 15 Arhitectura pe trei niveluri

5.5.2 Argumentare

Arhitectura client/server pe trei niveluri este utilizată atunci când noua aplicaţie client/server distribuită trebuie să permită (comparativ cu arhitectura pe două niveluri) performanţe mai bune, flexibilitate, mentenabilitate (întreţinere), reutilizabilitate, şi scalabilitate, în timp ce se ascunde complexitatea prelucrării distribuite faţă de utilizator. Aceste caracteristici fac din arhitectura client/server pe trei niveluri o alegere des întâlnită pentru aplicaţiile Internet şi pentru sistemele informatice axate pe reţea (net-centric information systems).

Beneficiile aduse de arhitectura client/server pe trei niveluri includ:

- scalabilitate
- flexibilitate

Scalabilitatea este îmbunătăţită deoarece codul server-ului şi bazele de date sunt separate (ele pot exista iniţial pe un singur calculator gazdă şi mai târziu pot fi separate). Mai multe servere de

aplicaţii pot comunica cu o bază de date centrală sau un server de aplicaţie poate deservi clienţii în timp ce accesează mai multe baze de date ca un sistem extins.

Flexibilitatea este îmbunătăţită deoarece clientul, server-ul şi sistemele de baze de date pot fi fiecare înlocuite fără să afecteze celelalte componente, furnizând interfeţe care de asemenea nu trebuiesc modificate.

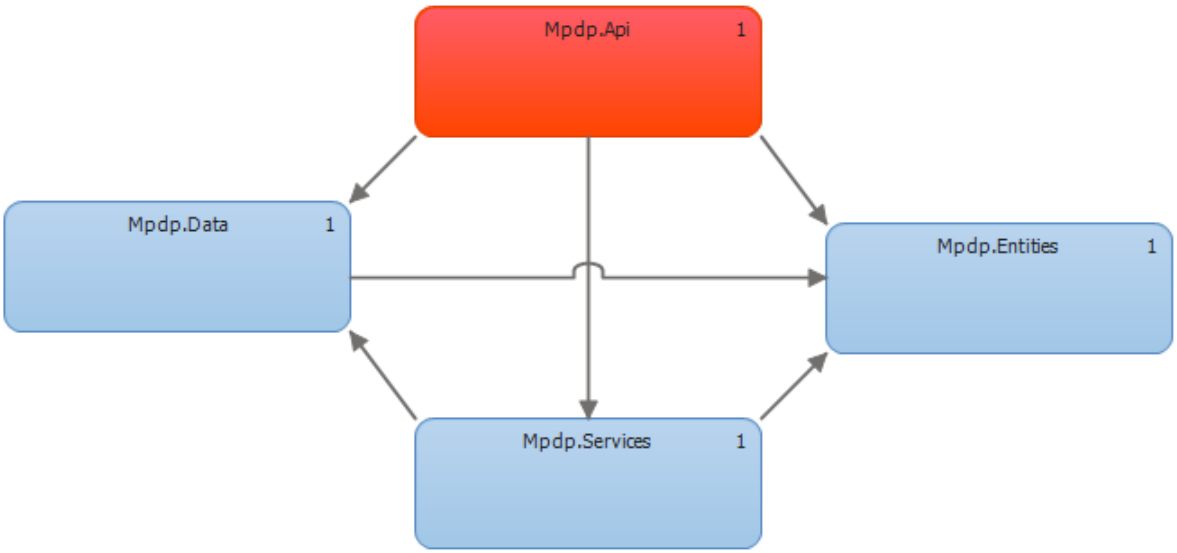
5.6 Prezentarea serviciului web ca "backend as a service"

Server side aplicația are la bază urmatoarele tehnologii:

1. ASP.NET Web API pentru accesarea datelor de către clienții web
2. Entitiy framework ca "*Object-relational Mapper*" pentru accesarea datolor (SQL Server)
3. Autofac pentru a rezolva dependentele către servicii şi pentru implementarea containerelor de tip IOC (Inversion Of Control)
4. Automapper pentru a mapping-ul dintre entitătile domeniului (Domain Entitites) si entițiile view-ului (ViewModel /MediaType)
5. FluentValidation pentru validarea entitațiilor view-ului în controller-ele API-ului

Următoarea figură descrie componentele din spatele serviciului expus, fiecare componentă in cadrul soluției fiind structurată pe un proiect diferit:

Fig. 16 componente serviciu expus



Cel mai jos nivel **Mpdp.Data** (My Personal Development Plan) ocupă rolul de Data Access Layer. Interactiunea cu baza de date fiind intreținută folosind Entity Framework.

Următorul nivel descris de componenta **Mpdp.Entities** definește toate modelele de care are nevoie aplicația My Personal Development Plan fiind in legatura 1:1 cu cele din baza de date.

La nivelul de business **MPDP.Services** se asigura doar un serviciu și anume cel de gestiune a utilizatorilor (MembershipService) care va fi ijectat în controllere-le API-ului unde este necesar.

Tot la nivel de business avem componenta Mpdp.API care va raspunde cererilor înaintate de client.

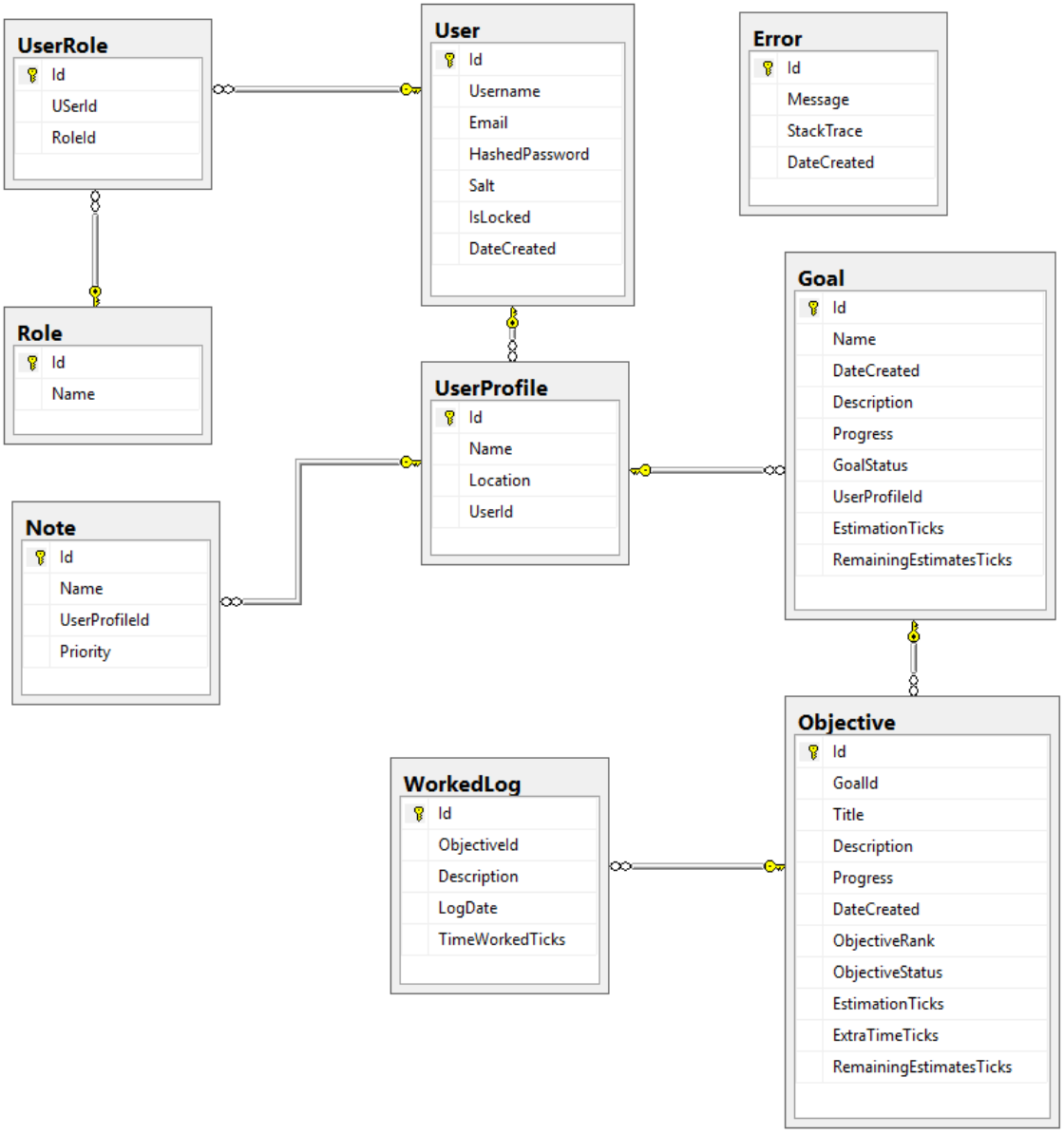


Fig. 17 Diagrama entității-relații

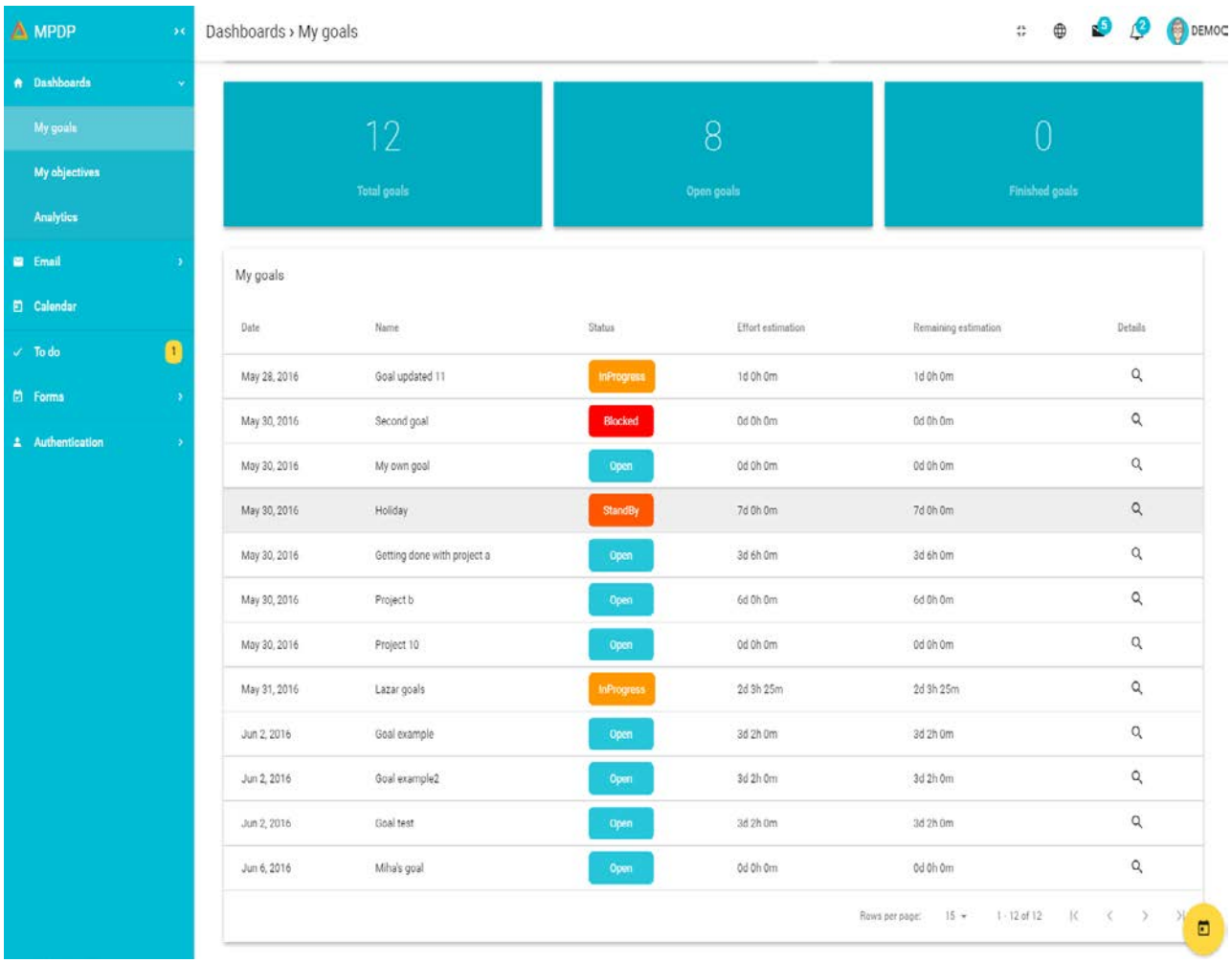
5.7 Prezentarea modulului grafic

Modulul grafic reprezintă partea de interfață cu utilizatorul al aplicației My Personal Development Plan denumit pe scurt client. Partea de client este implementată folosind tehnologia AngularJS versiunea 1.5 in combinație cu Material Design.

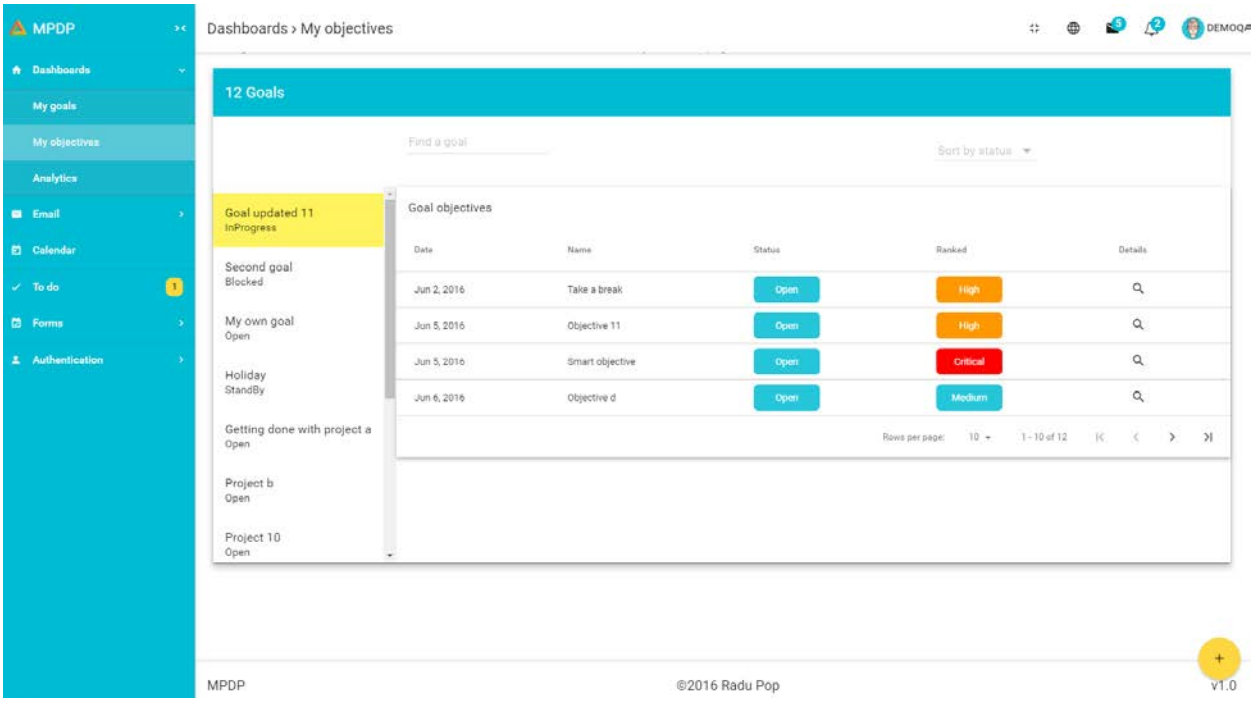
Aplicația este constuită folosind conceptul SPA(*Single Page Application*) propus de tehnologia Angular. Structura este propusă de către celebrul inginer de la Google John Papa.

5.8 Exemplificarea unor cadre de utilizare

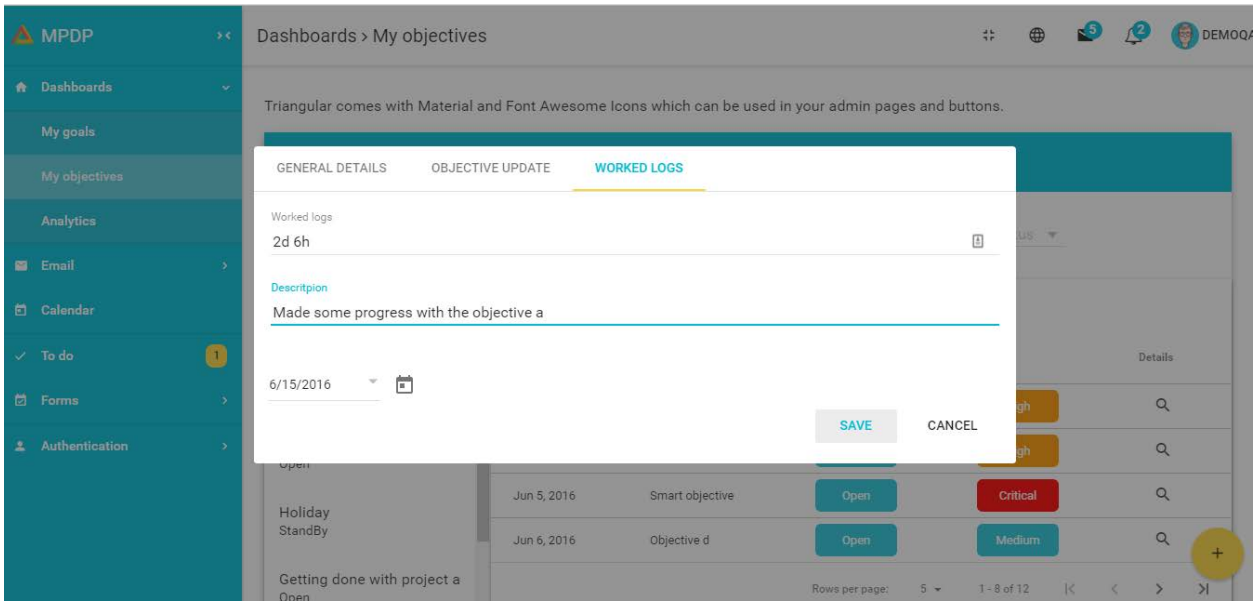
Vizualizarea scopurilor:



Vizualizarea obiectivelor :

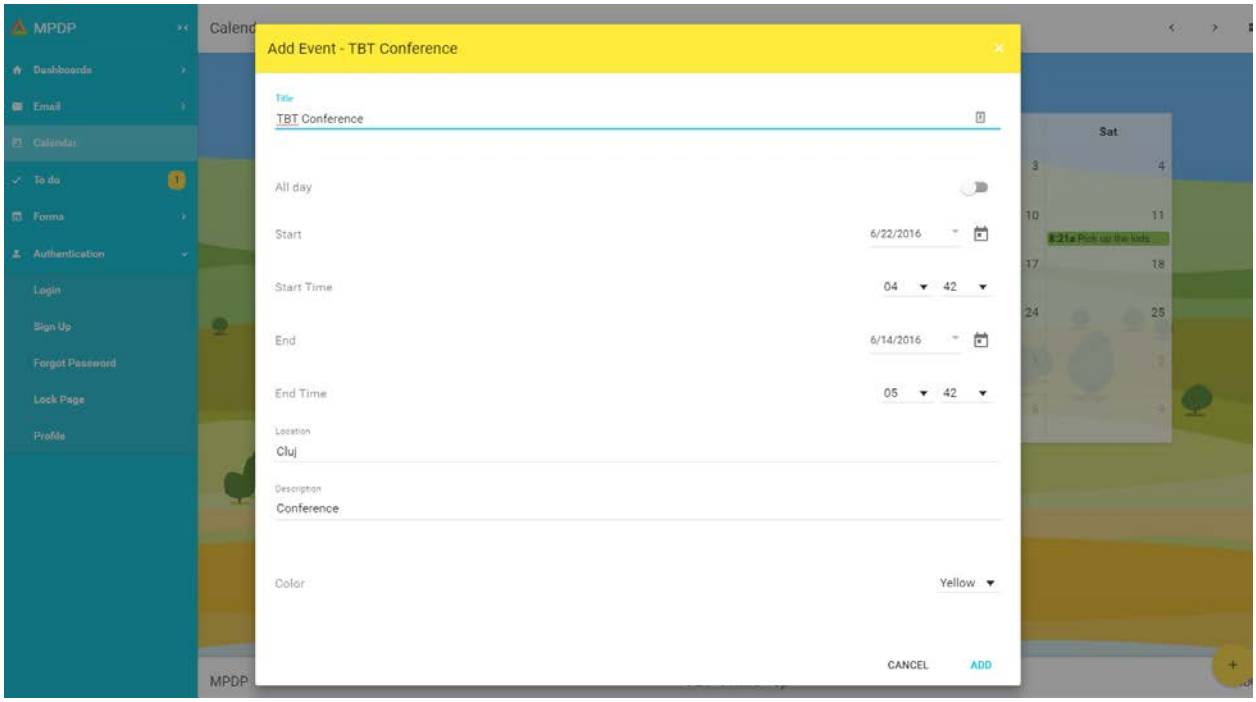


Înregistrarea efortului ca timp pe un anumit obiectiv:



Calendarul de evenimente:

Adaugarea unui nou eveniment:



6. Concluzii

Pornind de la o idee interesantă şi destul de comună mai precis gestiunea timpului ca şi element de succes în realizarea unor scopuri fixate, am încercat să modelez domeniul problemei încât să existe o gamă largă de functionalități dar totuşi înglobate doar pe un serviciu. Această abordare am decis să o justific așa cum am lăsat de înțeles la început doar acum, la finalul lucrării. Pentru a pune cât ma bine în valoare avantajele unei arhitecturi orientate pe servicii cred că este esențial mai întâi să observăm o necesitate obligatorie da a separa anumite funcționalități în vederea gestionării lor independent și o viziune clară în ceea ce presupune deschiderea infrastructurii create la a implementa noi funcționalități.

Consider că am asigurat o infrastructură care permite uşor extinderea și implementarea unor noi componente. Din punct de vedere comercial, aplicația poate fi livrată publicului larg și folosită în forma care este în prezent. Cu o interfață profesională și totodată simplă de folosit, asigură o experiență plăcută oricarui tip de utilizator.

Din perspectivă teoretică am încercat să ating toate aspectele necesare întelegerii acestui subiect printr-o prezentare a capitolelor care restrâng problema tot mai în detaliu pe măsura parcurgerii lor. Domeniul prezentat este unul amplu și necesită cunoștinte solide ale mediului de proeictare şi o întelegere a unor decizii arhitecturale care implică o abordare riguroasă însă ca punct de plecare noțiunile teoretice prezentate şi demonstrate practic în aplicația dezvoltată asigura o imagine mai clară

Rezultatul final al lucrării s-a transpus într-o aplicaţie care îndeplineşte obiectivele formulate iniţial adică gestionarea progresului prin parametrizarea scopurilor şi a obiectivelor ca fiind:

- ***S**pecifice*
- ***M**ăsurabile*
- ***A**ccesibile*
- ***R**ealistice*
- *încadrate în **T**imp*

Bibliografie

[1]There's a S.M.A.R.T. way to write management's goals and objectives

[2] *** Hypertext Transfer Protocol -- HTTP/1.1*w3c*

[3] *** *Building Web Apps with Go*

[4] *A mugwump's-eye view of Web work* Communications of the ACMPages 21-23

[5] *Web Engineering*2006

[6] *Servicii Web - Concepte de baza și implementari*POLIROM2006

[7] Glossary for the OASIS WebService Interactive Applications (WSIA/WSRP)

[8] *SOAP Web Services*2003

[9] Oracle GlassFish Server Message Queue 4.5 Developer's Guide for Java Clients*Oracle*

[10] *RESTful Web Services*O'Reilly2007

[11] *** http://restpatterns.org/

[12] ***http://rest.blueoxen.net

[13] JavaFX și comunicarea prin RESTful Web Services*Today Software Magazine*

[14] SOA, Web Services, and RESTful Systems*Drdobbs*http://www.drdobbs.com/web-development/soa-web-services-and-restful-systems/199902676

[15] Types of Web Services SOAP,XML-RPC and Restful*Php Flow*http://phpflow.com/php/web-service-types-soapxml-rpcrestful/

[16] *AngularJS: Up And Running*SeptemberO'Reilly Media2014

[17]*** http://mvvmlight.codeplex.com/

[18] *** https://erazerbrecht.wordpress.com/2015/10/13/mvvm-entityframework/

[19] Architectural Styles and the Design of Network-based Software Architectureshttps://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[20] *RESTful Web Services*O'Reilly Media2007

[21] Introduction to REST and .net Web APIhttps://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/

[22]***http://www.hanselman.com/blog/OneASPNETMakingJSONWebAPIsWithASPNETMVC4BetaAndASPNETWebAPI.aspx

[23] *** https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md