MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

ELEMENTE DE ANALIZĂ ŞI EVALUARE A SERVICIILOR WEB

LUCRARE DE LICENŢĂ

Absolvent: **Anca Loredana SUCIU**

Coordonator ştiinţific: **As. Ing. Cosmina IVAN**

2017

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Anca Loredana SUCIU**

ELEMENTE DE ANALIZĂ ȘI EVALUARE A SERVICIILOR WEB

1. **Enunţul temei:** *Lucrarea de față își propune analiza serviciilor Web din prisma performanței acestora, astfel încât, pornind de la un set de operații comune implementate atât folosind REST, cât și SOAP, se realizează teste de performanță în urma cărora se analizează și se compară metricile de performanță în cele două cazuri.*
2. **Conţinutul lucrării:** *Cuprins*, *Introducere*, *Obiectivele proiectului*, *Studiu Bibliografic, Analiză şi Fundamentare Teoretică, Proiectare de Detaliu şi Implementare, Testare, Validare şi Evaluare, Manual de Instalare şi Utilizare, Concluzii*, *Bibliografie*, *Anexe*
3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare
4. **Consultanţi**: **As. Ing. Cosmina IVAN**
5. **Data emiterii temei:** 1 martie 2017
6. **Data predării:** 8 Septembrie 2017

Absolvent: _____

Coordonator ştiinţific: _____

Cuprins

Capitolul 1. Introducere – Contextul proiectului

Scopul lucrării este de a evalua și de a compara performanțele serviciilor Web, implementate prin două modalități diferite: o implementare bazată pe SOAP (Simple Object Access Protocol), respectiv o implementare bazată pe REST (Representational State Transfer). Aceste două modalități prezintă abordări diferite în dezvoltarea aplicațiilor, fiind aplicate în funcție de cerințele și nevoile utilizatorilor.

SOAP reprezintă un cadru standardizat pentru construcția și procesarea mesajelor independent de capacitățile utilizatorului ce invocă mesajele, în timp ce REST constituie un set de principii ce se aplică pentru crearea serviciilor, având la bază protocolul HTTP. Deși sunt două tehnologii conceptual diferite, SOAP fiind un protocol, iar REST o arhitectură, ambele permit implementarea serviciilor web.

1.1. Contextul proiectului

Serviciile Web sunt foarte răspândite în diverse aplicații Web, fiind descrise ca și servicii oferite clienților prin intermediul internetului. Uneori serviciile Web sunt înțelese ca site-uri Web, dinstincția dintre ele fiind una importantă, și anume faptul că serviciile Web nu oferă utilizatorilor o interfață grafică GUI-Graphical User Interface, spre deosebire de aplicațiile Web. Mai concret, o aplicație Web (un site Web) reprezintă o colecție de mai multe pagini Web, iar fiecare pagină poate conține mai multe servicii Web.

Pentru crearea serviciilor Web există două metode care se utilizează (SOAP și REST), dar care diferă atât în procesarea datelor, precum și în modul în care oferă serviciile.

SOAP reprezintă un protocol standardizat definit de standardul W3C, folosit pentru transmiterea mesajelor de tip cerere sau răspuns. Transmiterea mesajelor se realizează folosindu-se formatul XML, atfel că datele sunt independente de platformă. Mesajele SOAP sunt transmise între aplicațiile de tip furnizor și aplicațiile de tip client, prin intermediul structurilor (envelops) SOAP.

REST e o arhitectură ce rulează folosind ca protocol de transport HTTP și asigură interacțiunile între clienți și servicii, având un număr limitat de operații. REST este o alternativă pentru SOAP, utilizând în locul cererilor de tip XML, URL-uri simple. Spre deosebire de SOAP, care oferă răspunsurile doar în format XML, REST oferă două formate pentru răspuns: XML și JSON.

După cum s-a menționat anterior, SOAP reprezintă un protocol, în timp ce REST este o arhitectură. Această distincție majoră între ele sugerează faptul că nu se compară tehnologiile între ele, ci se compară performanțele serviciilor Web, mai exact același set de servicii, implementate atât cu SOAP, cât și cu REST.

1.2. Cerințele proiectului

Având în vedere că se dorește analiza comparativă a serviciilor Web dezvoltate prin SOAP, respectiv REST, e nevoie de un set de operații comune ce vor fi implementate folosind alternativ ambele modalități amintite.

Pentru o cât mai reală comparație între performanțele obținute în cazul fiecărei tehnologii utilizate, este necesară respectarea următoarelor cerințe:

-utilizarea unui server comun ce va oferi serviciile menționate

-utilizarea unui set comun de operații, pentru a se compara servicii de aceeași complexitate

-utilizarea unor instanțe comune pentru baze de date, pentru a se păstra dimensiunea datelor

Având toate aceste caracteristici, se vor compara metricile de performanță rezultate în urma testării serviciilor ce urmează a fi prezentate.

1.3. Conținutul lucrării

Următoarele capitole vor prezenta procesul de dezvoltare al serviciilor, precum și modalitățile de analiză și studiu al performanțelor și, nu în ultimul rând, compararea rezultatelor obținute.

În capitolul 2 se vor prezenta obiectivele proiectului, care au dus la crearea contextului necesar ce a permis comparația între serviciile Web implementate atât cu SOAP, cât și cu REST.

În capitolul 3 se va face o scurtă introducere a conceptului de serviciu Web, iar apoi se vor prezenta informații obținute în urma studiului altor lucrări similare. De asemenea, se vor prezenta pe scurt conceptele de testare de servicii, testare de performanță, precum și instrumentele ce vor utilizate în studiul metricilor de performanță al serviciilor Web.

Capitolul 4 va descrie teoretic noțiunile folosite, ajutând la o mai bună înțelegere a acestora. Astfel, se va prezenta conceptul de serviciu Web, precum și modalitatea de funcționare, bazată pe principiul de client-server. De asemenea, se vor prezenta arhitectura SOA, protocolul SOAP împreună cu formatul mesajelor SOAP și elementele constitutive (WSDL,UDDI), precum și arhitectura REST, serviciile RESTful și protocolul HTTP, pe care se bazează. Se vor descrie, de asemenea, tehnologiile folosite in implementarea serviciilor web, precum si in analiza acestora.

În capitolul 5 se va prezenta modalitatea de implementare a serviciilor Web, constând în prezentarea bazei de date folosite, prezentarea proiectului, a cazurilor de utilizare, precum și a operațiilor implementate. De asemenea, vor fi prezentate modalitățile folosite pentru a valida serviciile și pentru a analiza performanțele acestora.

Capitolul 6 va prezenta testele funcționale realizate cu scopul de a valida corectitutidea serviciilor implementate, precum și analiza metricilor de performanță ce urmează a fi comparate.

Capitolul 7 va oferi un sprijin util în instalarea proiectului, atât privind serverul ce oferă serviciile, cât și instalarea clientului și înțelegerea modului cum acesta invocă serviciile Web.

Rezultatele obținute în urma analizei comparative a metricilor de performanță se vor prezenta în capitolul 8, care va conține, de asemenea, și posibilități de dezvoltare ulterioară.

Capitolul 2. Obiectivele proiectului

Principalul obiectiv al proiectului este de a compara performanța serviciilor Web, implementate prin două modalități diferite: folosind protocolul SOAP (Simple Object Access Protocol), respectiv arhitectura REST (Representational State Transfer). Din acest obiectiv principal derivă următoarele seturi de obiective secundare:

- **Crearea unui mini-context de aplicație care să permită evaluarea celor două posibile căi de implementare a serviciilor Web**

Acest obiectiv include stabilirea setului de servicii dorite, precum și implementarea acestora folosind cele două tehnologii amintite: SOAP și REST. Din punct de vedere funcțional, serviciile implementate vor putea fi folosite pentru managementul situației școlare a studenților din anii terminali, precum un catalog virtual conținând datele despre studenți, mediile obținute pe parcursul anilor, profesorii îndrumători, lucrările de licență și notele obținute în urma susținerii licenței. În acest scop, următoatele operații vor putea fi efectuate de către utilizatorul aplicației:

- Extragerea tuturor studenților înregistrați
- Înregistrarea unui nou student
- Găsirea unui student după numărul său matricol
- Ștergerea unui anumit student
- Vizualizarea studenților cu cele mai mari note obținute la susținerea licenței (spre exemplu, cu notele cuprinse între 8 și 10)
- Vizualizarea notelor la licență a studenților de la o anumită specializare
- Extragerea datelor tuturor studenților asignați unui anumit profesor
- Extragerea studenților asignați profesorilor dintr-un anumit departament
- Extragerea temelor de licență a tuturor studenților asignați profesorilor dintr-o anumită comisie

- **Identificarea și studiul instrumentelor utile pentru validarea și evaluarea performanțelor serviciilor**

După implementarea serviciilor Web amintite mai sus, e necesară validarea acestora, pentru a se verifica dacă funcționalitatea este cea dorită. În acest scop, se va alege un instrument ce permite testarea funcțională a serviciilor SOAP, precum și a serviciilor RESTful. De asemenea, se va alege un instrument ce oferă posibilitatea de a testa performanțele serviciilor, urmând ca acestea să fie analizate și comparate.

- **Stabilirea scenariilor de test în condițiile în care se evaluează serviciile**

Setul de servicii prezentate mai sus vor fi implementate atât în SOAP, cât și în REST, la nivel de server, care va furniza serviciile. Clienții vor fi simulați folosind instrumentele de testare alese, astfel încât aplicația să poată fi testată cu un "workload" mare, corespunzător unor condiții de stres, pentru a se permite analiza comportamentului serverului atât în cazul serviciilor SOAP, cât și a serviciilor RESTful.

- **Stabilirea metricilor de performanță ce vor fi evaluate**

Timpul de răspuns minim/maxim, average-ul, throughput-ul, precum și utilizarea resurselor, memoria utilizată și CPU-ul vor fi calculate pentru serviciile mai “critice”, care ar solicita serverul cel mai tare tare, încercând chiar să-i găsească punctele slabe (condițiile în care comportamentul serverului nu mai este cel așteptat).

- **Analiza rezultatelor obținute în urma testelor de performanță**

Având calculate toate metricile menționate mai sus, se vor genera grafice și se vor analiza rezultatele, urmând ca în urma comparației să se observe care dintre sevicii este mai eficient și în ce mod. Pentru serviciile care solicită serverul mai tare, se vor efectua mai multe rulări ale testelor de performanță, rezultatele fiind memorate în fișiere și comparate ulterior.

Capitolul 3. Studiu Bibliografic

3.1. Studii bibliografice ce vizează serviciile Web

Serviciile Web, privite ca servicii oferite clienților de către un furnizor, pe principiul de client-server, au început să prindă contur aproximativ prin anii 1999-2000, ajungând să fie extrem de folosite în aplicațiile Web actuale, al căror număr este în creștere, acestea fiind mereu într-o continuă dezvoltare.

Cele două modalități de dezvoltare a serviciilor Web, respectiv SOAP și REST, diferă mult atât din punct de vedere contextual, cât și al utilizării, deși ambele au scop comun, ajutând la implementarea serviciilor Web. SOAP reprezintă un protocol bine dezvoltat și utilizat în industria Web, fiind standardizat de W3C (World Wide Web Consortium). REST este o arhitectură, prezentată pentru prima data de Dr. Roy Thomas Fielding în anul 2000, o arhitectură ce a ajuns să câștige o mare popularitate datorită simplității, scalabilității. Renumite companii utilizează REST, printre care Google sau Amazon. Totuși, există situații când implementarea serviciilor folosind SOAP este mai potrivită, aceasta având un sistem de securitate integrat și oferind posibilități de construcție a unor operații mai complexe.

În ceea ce privește definiția unui serviciu Web, aceasta poate fi interpretabilă în funcție de aplicabilitatea serviciului, neexistând o definiție unanim acceptată a acestui concept.

Astfel, W3C (World Wide Web Consortium) caracterizează un serviciu Web ca fiind un sistem software proiectat pentru a suporta interoperabilitatea interacțiunilor de la o mașină la alta printr-o rețea. IBM consideră un serviciu Web ca fiind o aplicație modulară bazată pe tehnologii de Internet, menită să îndeplinească activități specifice și conformându-se unui format tehnic specific. Conform Microsoft, serviciile reprezintă partea de procesare a datelor unei aplicații, disponibilă la nivel de program și beneficiind de funcționalitățile oferite de Internet. După Sun, un serviciu Web e o aplicație care acceptă cereri din partea altor sisteme dispuse în Internet sau Intranet, mediate de tehnologii de comunicație neutre și agile.[12]

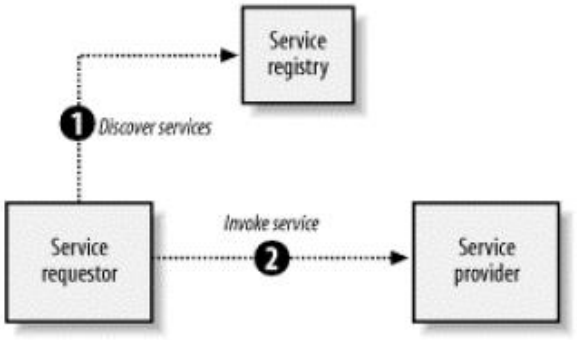
În sursa [1], autorul Ethan Cerami afirmă că serviciile Web oferă o paradigmă evoluată pentru construcția aplicațiilor web distribuite. De asemenea, el descrie serviciul Web ca fiind un serviciu valabil prin intermediul Internetului și care folosește un sistem de schimb de mesaje bazat pe XML, nefiind legat de niciun sistem de operare sau de vreun limbaj de programare. Autorul prezintă și arhitectura unui serviciu Web, care poate fi privită din două perspective diferite: prin prisma rolurilor sau prin prisma stivei de protocoale ale serviciilor.

În ceea ce privește rolurile prezente în serviciile Web, sunt prezentate:

-*Furnizorul de servicii*, care asigură valabilitatea serviciului prin Internet

-*Clientul serviciului*, care deschide o conexiune invocând serviciul printr-o cerere specifică

-*Registrul de servicii*, reprezentând un loc central unde toți furnizorii de servicii pot publica servicii noi sau le pot găsi pe cele deja existente



Figură 3-1 Rolurile serviciului Web

Referitor la stiva de protocoale, acestea sunt:

-*Protocolul de transport al serviciului*, incluzând HTTP (Hypertext Transfer Protocol), SMTP(Simple Mail Transfer Protocol) sau FTP(File Transfer Protocol)

-*Nivelul de transmitere al mesajelor XML*, folosit cu precădere în SOAP sau XML-RPC

-*Descrierea serviciului*, realizată prin WSDL (Limbajul de Descrierere a Serviciului Web)

-*Descoperirea serviciului*, nivel responsabil pentru centralizarea serviciilor într-un loc comun

Figură 3-2 Stiva de protocoale a serviciilor [1]

Una dintre lucrările de referință pentru a ilustra diferențele principale între SOAP și REST o constituie lucrarea [13]. Una dintre principalele diferențe amintite în această lucrare este că SOAP este proiectat având componente strâns cuplate, similar cu RPC (Remote Procedure Call), în timp ce componentele REST sunt slab suplate, fiind similar cu navigarea pe link-uri Web. Autorii prezintă atât avantajele, cât și dezavantajele abordărilor SOAP, respectiv REST. Astfel, SOAP are avantajul de a fi strâns cuplat, în timp ce avantajul oferit de REST este scalabilitatea și modalitatea ușoară de acces la operații. Dezavantajul arhitecturii REST este că necesită un număr mare de obiecte pentru spațiul de nume, iar dezavantajul SOAP este că necesită porturi dedicate ale clienților pentru diferite tipuri de notificări.

Referitor la analiza perfomanțelor și a calității serviciilor Web, sursa [2] prezintă un model clar de calcul al calității serviciilor, atât pentru utilizatori, cât și pentru furnizorii de servicii Web. Experimentele lor s-au bazat pe calculul timpului de răspuns și a costului serviciilor Web, invocate simultan de utilizatori multipli.

O altă lucrare în care s-au analizat performanțele serviciilor Web este lucrarea [3]. Experimentele s-au bazat pe dezvoltarea unui model EtE (end-to-end) pentru a monitoriza performanțele unui site Web. Rezultatele au fost ilustrate prin grafice generate pentru trei sudii de caz diferite a trei servicii Web distincte. Autorul se concentrează pe timpul de răspuns, durată, dimensiune și timpul de execuție pentru fișiere de dimensiuni largi.

O resursă foarte utilă privind comparatia între SOAP și REST o reprezintă lucrarea [4]. Autorul acestei lucrări a studiat comparativ serviciile Web, implementând operațiile elementare GET, PUT, POST și DELETE, baza de date conținând o singură tabelă. Implementarea a fost realizată alternativ folosind SOAP și REST, urmând să fie calculat timpul de răspuns în fiecare caz. În urma comparației s-a constatat ca REST este mai performant ca SOAP, având un timp de răspuns mai bun.

3.2. Studiul instrumentelor pentru testarea și validarea serviciilor

3.2.1. Instrumente ce vizează validarea serviciilor web

Urmărind studiul comparativ al serviciilor Web, implementate cu SOAP, respectiv cu REST, a fost necesară căutarea unor instrumente potrivite pentru testarea funcțională a serviciilor, precum și pentru testarea performanței acestora. De asemenea, un studiu minimal al metricilor de performanță a fost util în crearea testelor de performanță și a comparării performanțelor celor două tipuri de servicii.

Testarea de servicii web se poate realiza în trei moduri diferite, și anume: fie se testează manual serviciile (de exemplu, în browser pentru RESTful), fie se creează cod pentru testarea acestora (de exemplu: REST Assured API, care funcționează doar în cazul REST), fie se folosesc instrumente special concepute, care asigură posibilitatea de a se valida corectitudinea serviciilor.

În lucrarea “Automating and testing a REST API ”, autorul Alan Richardson descrie conceptul de API (Application Programming Interface), acesta reprezentând o interfață a unei aplicații, proiectată spre a fi utilizată de alte calculatoare, care vor invoca servicii Web.De asemenea, autorul se concentrează pe API-uri HTTP, explicând concret conceptele de cerere, respectiv răspuns HTTP.Se prezintă astfel și operațiile GET, PUT, POST, DELETE și codurile statusurilor HTTP. Toate aceste elemente vor fi detaliate în următorul capitol. [5] Cartea prezintă, astfel, doar servicii RESTful. Autorul sugerează un set de instrumente ce oferă ajutor în testarea API-urilor, precum: cURL,Postman sau REST Assured.

-**cURL** [5] constă, practic, în introducerea unor comenzi în linia de comandă, prin intermediul cărora se vor trimite cereri Http cu scopul de a se valida buna funcționare a serviciilor. Acest instrument oferă posibilitatea de a observa și de a manipula cererile, ajutând la înțelegerea acestora și totodată, la deprinderea utilizării lor corecte. Totuși, cURL nu constituie un instrument complex pentru a testa API-uri, ci doar contribuie la înțelegerea mai bună a modului cum pot fi testate, prin faptul că validează corectitudinea cererilor, care se vor putea utiliza mai apoi în teste mai complexe.

-**Postman GUI** [5] e un instrument ce e foarte simplu de utilizat, datorită faptului că oferă o interfață pretenoasă utilizatorilor. Postman este o extensie a aplicației Chrome, fiind ulterior convertit într-o aplicație desktop. Ca și elemente, Postman conține

posibilitatea de a alege metoda dorită (GET, PUT, și așa mai departe), precum și introducerea URL-ului. De asemenea, conține un buton denumit "Params" care oferă o variantă simplă de a edita parametrii doriți în cerere. În urma trimiterii cererii, se va primi un răspuns vizualizat în format JSON sau XML, în funcție de modul în care au fost implementate serviciile RESTful.

-**REST Assured API** [5] reprezintă o modalitate de testare automată, folosind cod Java. Instalarea acestuia este una simplă și reprezentând un proiect Maven, implică doar adăugarea dependinței pentru *rest-assured* în fișierul pom.xml:

```
<dependency>
<groupId>io.rest-assured</groupId>
<artifactId>rest-assured</artifactId>
<version>3.0.1</version>
</dependency>
```

Următorul pas este de a implementa efectiv scenariul de test, folosind adnotări specifice, precum în exemplu următor:

```
@Test
public void findAllStudents () {
given ().
contentType("text/json").
when ().
get("http://localhost:8080/students/all-students").
then ().
statusCode (200);
}
```

Această metodă de a valida corectitudinea serviciilor implementate este una utilă, dar un mare dezavantaj al acesteia îl constituie faptul că metoda este valabilă doar pentru serviciile implementate folosind REST, iar astfel nu permite validarea serviciilor Soap și RESTful în paralel.

Urmărind o comparație clară între serviciile Web implementate atât cu REST, cât și cu SOAP, este necesar un instrument ce oferă posibilitatea testării ambelor tehnologii în parte.

-**WebInject** [14] e un instrument gratuity care se folosește pentru testarea aplicațiilor Web și a serviciilor Web, fiind util atât pentru SOAP, cât și pentru REST. Instrumentul este scris în Perl și poate fi instalat pe MS Windows, GNU/Linux, BSD, Solaris sau MAC OS. Dezavantajul acestui instrument e că funcționează în linia de comandă, neavând interfață grafică.

-**SoapUI**, un alt instrument folosit pentru testarea funcțională a serviciilor Web implementate cu SOAP sau REST este SoapUI, prezentat de către compania SmartBear în diverse articole, precum și pe site-ul oficial [6]. SoapUI este un instrument folosit la nivel mondial, ce oferă suport în testarea funcțională de servicii web de tip SOAP, în testarea funcțională de API-uri REST, aserții de mesaje utile în validarea răspunsurilor, precum și posibilitatea de a refactoriza testele. Creat în anul 2006, acest instrument de testare a ajuns să fie utilizat de o multitudine de utilizatori din întreaga lume, iar faptul că beneficiază de gratuitate constituie un mare avantaj pentru utilizatorii acestuia.

SoapUI asigură suport atât pentru testarea serviciilor implementate cu SOAP, cât și pentru cele implementate cu REST. Astfel, la crearea proiectului exista două opțiuni

diferite (Crearea unui proiect SOAP sau a unuia REST). Testele sunt organizate în proiecte, astfel că fiecare proiect poate avea mai multe suite de test, iar fiecare suita poate avea mai multe cazuri de test (Test cases). Acestea sunt create din mai mulți pași de test (test steps). Printr-un astfel de pas de test se organizează logica de validare a unei anumite funcționalități dintr-un serviciu web. O tehnică recomandată în crearea testelor funcționale este metoda cascadă, care validează operațiile pe rând, în ordinea în care acestea au fost create și executate. De asemenea, instrumentul oferă multiple validări: se validează conținutul răspunsului, sau dacă data întoarsă trebuie sa respecte un anumit timp, dacă se trimite un cod de eroare, precum și mesajul afișat în acest caz.

3.2.2. Instrumente ce vizează testarea performanței serviilor Web

Întrucât se urmărește compararea performanțelor serviciilor Web, este necesar studiul și înțelegerea conceptului de performanță în contextul de interes, respectiv a abordărilor ce orferă soluții pentru testarea performanței.

Conceptul de *performanță* a unui sistem software se referă la comportamentul sistemului în condiții normale de utilizare, raportându-se la parametri precum timpul de răspuns sau resursele utilizate.

Testarea de performanță reprezintă o metodă de testare non-funcțională și nu are ca scop găsirea de defecte în sistem, ci monitorizarea metricilor de performanță astfel încât să corespundă necesităților utilizatorilor. (Spre exemplu, se dorește ca timpul de răspuns să nu depășească 1s, caz în care se va raporta o eroare). Astfel, în cazul în care se va observa degradarea performanței, se vor putea îmbunătăți anumite aspecte, pentru a crește calitatea produsului.

Există mai multe subtipuri de teste de performanță, și anume: **teste de stres** (cu scopul de a verifica dacă sistemul se comportă acceptabil chiar și în cele mai rele condiții), teste de încărcare și stabilitate (asigură stabilitatea sistemului pe o perioadă lungă de timp cu încărcare completă – load&scalability), **teste de fiabilitate** (au ca scop măsurarea capacității sistemului de a rămâne operational pentru perioade lungi de timp – endurance), **teste de volum** (se testează performanțele sistemului pentru diferite volume ale bazei de date), **teste de capacitate** (se găsește punctul de usaj cel mai mare pentru ca sistemul se comportă acceptabil) sau **teste peak-spike** (se analizează sistemul pentru o încărcare foarte mare pe o perioadă scurtă de timp).

Testarea de performanță are astfel trei parametri esențiali ce trebuie monitorizați, privind metricile de performanță:

Timpul de răspuns (reprezintă intervalul de timp din momentul când un utilizator trimite o cerere până când acesta va primi răspunsul din partea serverului)

Utilizarea resurselor (se referă la memoria utilizată, CPU-Central Processing Unit, intrările/ieșirile bazei de date, tranzacții și așa mai departe)

Degradarea performanței (monitorizarea performanțelor și analiza acestora duce la observarea eventualelor degradări, caz în care se vor aduce imbunătățiri în acea zonă)

Pentru analiza principalelor metrici de performanță a căror comparație se va realiza pentru serviciile SOAP și REST, este necesar un instrument potrivit care ajută la calculul acestora, mai concret un instrument pentru realizarea testării performanței.

-LoadUI NG Pro [11] – reprezintă un instrument util în crearea unor teste complexe de performanță, fiind totodată ușor de utilizat. Intrumentul poate fi utilizat pentru REST, SOAP, JMS, MQTT sau alte formate API, fiind potrivit pentru comparația între SOAP și REST. Deoarece testele sunt ușor de creat, instrumentul permite utilizatorului să se concentreze mai mult de partea de analiză și nu pe crearea efectivă a testelor.

-LoadView [11]**–** este un instrument ce permite crearea de teste de performanță pentru încărcare și stabilitate. Spre deosebire de alte instrumente, LoadView oferă posibilitatea de a rula testele de performanță pe browsere reale, asigurând astfel acuratețea datelor. Este potrivit în special pentru teste realizate pentru aplicații Web.

-LoadComplete [11]–reprezintă un instrument comforabil și ușor de folosit în crearea testelor de performanță. Acesta permite crearea și executarea unor teste realistice de performanță pentru site-uri web si pentru aplicații web. Acesta creează automat teste realistice de încărcare prin înregistrarea interacțiunilor dintre utilizatori și simularea a sute de utilizatori simultani, atât local, cât și în cloud. Pentru a putea fi instalat, acesta necesită un sistem de operare de 64 biți (Windows XP professional, Windows 7, etc)

Apache JMeter [24] este un instrument gratuit proiectat pentru realizare testelor de încărcare și studiu al performanțelor, după cum este descries de către Dmitri Tikhansi în articolul “Getting started with Apache JMeter” din DZone. Acest instrument este o aplicație desktop implementată în Java, care rulează pe Mac, Linux și Windows și oferă posibilitatea de a simula mai mulți utilizatori simultani pentru a testa performanța unui sistem în anumite condiții.

JMeter poate fi utilizat atât în linia de comandă, cât și prin intermediul unei interfețe grafice, foarte utilă pentru utilizatori. Aplicația funcționează pentru mai multe tipuri de servere și protocoale, incluzând: Web (HTTP și HTTPS), SOAP/REST, FTP, TCP, SMTP, Baze de date cu JDBC sau MongoDB (bază de date non-relațională).

Instrumentul este foarte ușor de instalat. Proiectele de test sunt grupate in planuri de test (Test plan). În funcție de ceea ce se dorește a fi testat, se pot alege diverse elemente ce vor fi conținute în planul de test. Printre cele mai utile astfel de elemente, se numără: Thread Group (specifică numărul de utilizatori simulați), Timers (se setează durata intervalului de timp dintre request-uri), Configuration Elements, Samplers (elemente ce generează cererile clienților, precum HTTP Request sau SOAP/XML-RPC Request), Aserții (utilizate pentru validarea conținutului unui răspuns) și Listeners (oferă diverse posibilități de vizualizare a rezultatelor, precum tabele sau grafice).

Metricile de performanță pe care Apache JMeter le oferă sunt prezentate și descrise în tabelul următor:

Tabel 3-1 Metricile de performanță oferite de JMeter

Label (eticheta)	Reprezintă eticheta unei probe(sample). Se permite ca etichetele identice aparținând unor grupuri de thread-uri diferite să fie colecționate separate
#Samples (Probe)	Reprezintă numărul de probe care au aceeași etichetă
Average (media)	Timpul de răspuns mediu pentru un anumit set de rezultate
Median	Timpul obținut în mijlocul unui set de rezultate (50% din eșatioane nu au durat mai mult decât acest timp, iar restul au durat cel puțin cât medianul)
90% Line	90% dintre eșantioane au luat cel mult tot atâta cât această valoare, în timp ce restul de 10% au durat și mai mult
Min	Timpul de răspuns cel mai mic al eșantionelor cu aceeași etichetă
Max	Timpul de răspuns cel mai mare al eșantionelor cu aceeași etichetă
Error %	Procentul cererilor pentru care s-au obținut erori
Throughput (Lățime de bandă)	Se măsoară în cereri per secundă/minut/oră. Unitatea de timp este aleasă astfel încâr rata de afișare este cel puțin 1.0. Când throughput-ul este salvat într-un fișier CSV, se exprimă în cereri/secundă (ex: 30 cereri/minut se va salva ca fiind 0.5)
KB/Sec	Exprimă throughput-ul măsurat în Kilobytes per secundă.

Capitolul 4. Analiză şi Fundamentare Teoretică

4.1. Tehnologii utilizate în implementarea serviciilor web

4.1.1. Java Enterprise Edition (Java EE)

Java Enterprise Edition [14], cunoscută și ca J2EE, reprezintă o platformă utilizată pentru dezvoltarea și implementarea aplicațiilor și serviciilor Web, precum și a aplicațiilor distribuite, multi-platformă, bazate pe component sau aplicații pe mai multe nivele. Această platformă are la bază limbajul de programare Java, care folosește paradigmele programării orientate pe obiect.

Java EE extinte platforma Java SE (Standard Edition), oferind un API (Application Programming Interface) pentru servicii Web, precum și pentru arhitecturi distribuite și multinivel.

Platforma cuprinde un design bazat pe component modulare ce rulează pe serverul aplicației. De asemenea, platforma pune accentual pe convenție și adnotări pentru configurare.

Printre avantajele platformei Java Enterprise Edition, se numără: portabilitatea, scalabilitatea, interoperabilitatea, performanța și simplitatea. De asemenea, costurile reduse asigură un avantaj al acestei platforme.

Java EE e definită prin specificații, incluzând multiple specificații API, precum RMI, e-mail, JMS, servicii web, XML. Platforma fiind bazată pe component modulare, asigură programatorilor Java posibilitatea să creeze aplicații portabile și scalabile. Un server de aplicație Java EE se ocupă de tranzacții, securitate, scalabilitate, concurență și management al componentelor implementate, astfel permițând dezvoltatorilor software să se concentreze pe logica de business a componentelor.

4.1.2. Eclipse IDE

Eclipse [15] reprezintă un mediu de dezvoltare software, denumit în limba engleză “Integrated Development Environment” (IDE), care e scris preponderent în Java, fiind de asemenea folosit, cel mai frecvent, în implementarea aplicațiilor Java. Folosind anumite plugin-uri (precum Ada, ABAP, C, C++, C#, COBOL, Fortran, Javascript), Eclipse oferă și posibilitatatea de a crea aplicații în alte limbaje de programare.

Eclipse asigură dezvoltarea aplicațiilor folosind servere precum Tomcat, GlassFish și multe altele și este capabil să instaleze serverul necesar direct din IDE.

4.1.3. Apache Tomcat

Apache Tomcat [17], denumit în trecut Jakarta Tomcat, este un server web open-source și un container servlet dezvoltat de Apache Software Foundation (ASF). Apache Tomcat este un server HTTP care oferă funcționalitatea de container pentru Java Servlets, având o arhitectură modulară.

Serverul Apache Tomcat operează conform modelului cerere-răspuns. O cerere este transmisă în momentul în care în navigator este solicitată o pagină Internet

disponibilă în contextul serverului. Pe baza execuției clasei Java Servlet corespunzătoare, se generează un răspuns ce constă într-un document HTML. În mod implicit, Apache Tomcat defineşte o gazdă `localhost` şi un director de bază `webapps` unde vor fi dezvoltate toate aplicaţiile.

4.1.4. Apache Maven

Apache Maven [18] este o unealtă de construcție automata pentru proiectele Java. Ea abordează două aspecte importante: în primul rând, descrie modul în care este construit software-ul, iar în al doilea rând, descrie dependințele acestuia. Contrar instrumentelor anterioare, precum Apache Ant, Maven utilizează convențiile pentru procedura de construire și doar excepțiile trebuie scrise. Un fișier XML (spre exemplu fișierul pom.xml) descrie proiectul software construit, dependențele acestuia din alte module, precum și componentele externe, directoarele și plugin-urile necesare. Acesta vine cu obiective predefinite pentru a efectua anumite sarcini bine definite cum ar fi compilarea codului sau împachetarea acestuia.

Maven descarcă în mod dynamic bibliotecile Java și plugin-urile Maven din unul sau mai multe depozite, stocându-le într-un cache local. Acest cache local de artefacte descărcate poate fi actualizat, de asemenea, cu artefacte create de proiecte locale.

4.1.5. JAX-WS

JAX-WS (Java API for XML Web Services) [21] este o tehnologie Java folosită pentru dezvoltarea de servicii web și de aplicații ce le invocă, iar comunicația lor se bazează peXML. Folosind JAX-WS, se pot dezvolta servicii web orientate pe mesaje sau pe apeluri la distanță (RPC sau Remote Procedure Call). Invocarea unui serviciu web se realizează folosindu-se protocolul SOAP, acesta fiind bazat pe fișiere în format XML. Specificația SOAP conține structura antetelor, regulile de codificare, convențiile pentru reprezentarea cererilor, precum și a răspunsurilor.

Mesajele de tip SOAP sunt foarte complexe, însă acest lucru este ascuns utilizatorilor prin API-ul JAX-WS, care este inclus în proiect prin specificarea acestuia ca dependință în fișierul pom.xml.

Operațiile serviciul web se definesc în cadrul unei interfețe, urmând a fi implementate. Programatorul nu are nevoie să genereze sau să analizeze mesaje de tip SOAP, având în vedere că sistemul de execuție JAX-WS convertește cererile și răspunsurile în mesaje SOAP.

Un serviciu web implementat folosind JAX-WS este, practic, o clasă Java ce va fi adnotată cu @WebService (*javax.jws.WebService*), fiind astfel desemnată ca entitatea ce oferă serviciul web respectiv. Astfel, structura prin care se descrie un serviciu web (SEI- Service Endpoint Interface / Implementation) e o clasă sau o interfață Java unde vor fi declarate metodele ce vor putea fi invocate de către clienți. Aceste metode for fi specificate în cadrul interfeței, fiind adnotate folosind adnotarea @WebMethod, urmând să fie implementate în cadrul clasei ce va oferi serviciul web.

Figură 4-1 Arhitectura sistemului JAX-WS

4.1.6. JAXB

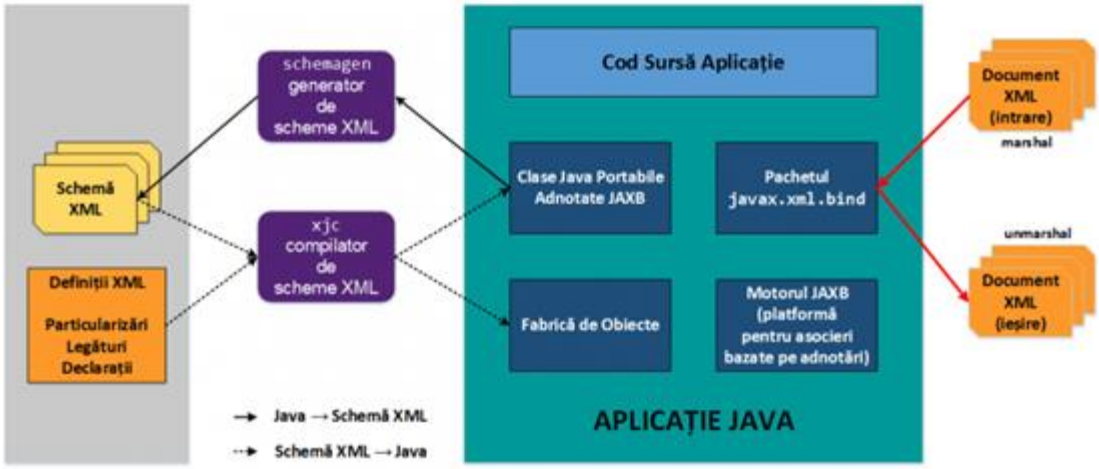
JAXB (Java Architecture for XML Binding) [22] asigură conversia între aplicațiile implementate în limbajul Java și schemele XML. Acesta poate fi folosit fie în combinație cu JAX-WS, fie independent, în lucrarea prezentă fiind folosite împreună. Astfel, *JAX-WS* deleagă către *JAXB* o mapare a tipurilor prezente în limbajul de programare Java, cu definiții XML. Programatorii nu trebuie să cunoască aceste detalii, ci doar să știe că nu orice clasă din limbajul de programare Java poate fi folosite ca și parametru sau valoare JAX-WS, având în vedere că Java oferă un set de tipuri de date mult mai complex decât schemele XML.

JAXB oferă un utilitar pentru compilarea unei scheme XML într-o clasă Java (denumit *xjc*), un cadru de rulare, precum și un utilitar de generare a schemei XML dintr-o clasă Java (denumit *schemagen*). Folosind utilitarul xjx, se poate porni de la o definiție a unei scheme XML (XSD – XML Schema Definition) creându-se obiecte Java corespunzătoare. Folosind utilitarul schemagen, se pornește de la obiectele Java, creându-se definițiile XSD.

După stabilirea corespondenţei dintre schema XML şi clasele Java, conversia între ele este realizată în mod automat prin intermediul JAXB. Informaţiile reţinute în cadrul documentelor XML pot fi accesate fără a fi necesară înţelegerea structurii lor. Clasele Java rezultate pot fi folosite pentru accesarea serviciului web dezvoltat. Adnotările conţin toate datele necesare pentru ca JAXB să realizeze conversia dintr-un format într-altul, astfel încât obiectele să poată fi transmise prin infrastructura de comunicaţie. JAXB asigură împachetarea obiectelor Java spre documente XML şi despachetarea documentelor XML în instanţe ale claselor Java.

Totodată, JAXB poate oferi validarea documentelor XML conform schemei generate, astfel încât acestea respectă constrângerile care au fost stabilite.

Prin urmare, *JAXB* reprezintă tehnologia standard utilizată pentru conversia datelor în cadrul serviciilor web de dimensiuni “mari” implementate prin *JAX-WS*. Totuşi, *JAXB* poate fi folosit şi independent de *JAX-WS*, atunci când se doreşte gestiunea conversiei dintre documente XML şi obiecte Java în cadrul unor aplicaţii informatice.



Figură 4-2 Schema conceptuală pentru JAXB

4.1.7. JPA (Java Persistence API)

Java Persistence API (JPA) [19] este o specificație Java pentru accesarea, persistența și gestionarea datelor între obiectele, respectiv clasele Java și o bază de date relațională (de exemplu MySQL). JPA este acum considerată abordarea standard a industriei pentru Object-Relational Mapping (ORM) în industria Java. JPA reprezintă o colecție de clase (scrise folosind limbajul de programare Java) ce oferă funcționalități legate de interogarea eficientă a informațiilor aflate în bazele de date relaționale. JPA este cu atât mai utilă cu cât volumul datelor este unul considerabil.

JPA e o interfață de programare ce implementează un nivel de abstractizare astfel încât toate informațiile sunt sunt stocate la nivel de obiecte, realizându-se o mapare între datele prezentate relational în baza de date, respectiv datele reprezentate sub formă de obiecte. Astfel, programatorul nu este nevoit să știe detalii despre informațiile existente în baza de date, el accesându-le direct la nivel de obiect.

Desigur, JPA este doar o specificație, nu un produs. Nu poate face persistență în sine. JPA este doar un set de interfețe și necesită o implementare. În prezent, pe piață există numeroase implementări JPA (unele comerciale, altele open-source) care oferă suportul necesar utilizării. De asemenea, JPA necesită o bază de date care să fie persistentă.

JPA permite maparea obiectelor obiect-relaționale să fie definite prin adnotări standard sau XML care descriu modul în care clasa Java se mapează cu o tabelă de bază de date relațională. De asemenea, JPA definește un EntityManager API pentru rularea interogărilor și tranzacționarea obiectelor în baza de date.

4.1.8. Spring Data JPA

Spring Data JPA [20] este un framework ce extinde JPA (Java Persistence API) prin adăugarea unui nivel suplimentar de abstractizare în partea de sus a furnizorului JPA. Acest nivel asigură suportul necesar pentru crearea de depozite (“repositories”) JPA prin extinderea interfețelor depozitelor Spring JPA.

Pașii realizați în folosirea framework-ului Spring Data JPA sunt:

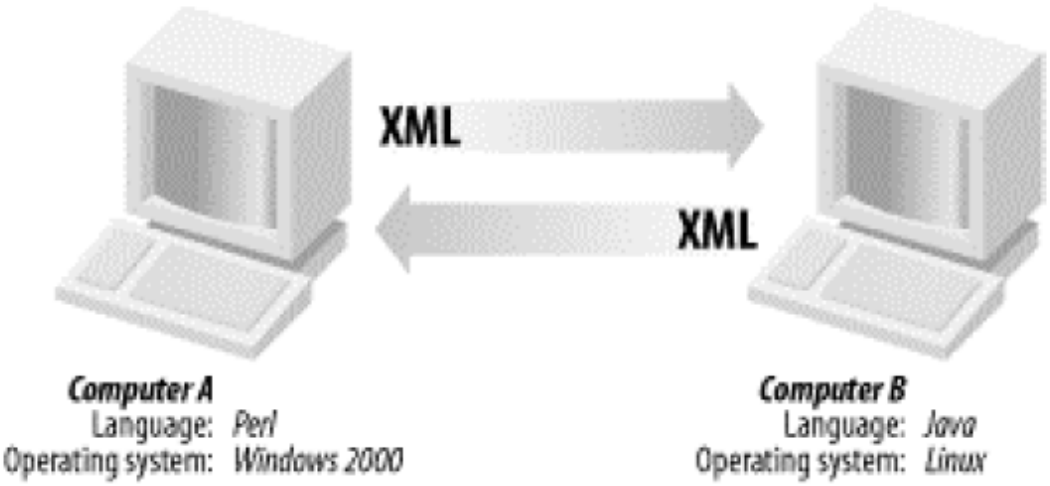
- Crearea entităților, adnotate folosind *@Entity* (O entitate reprezintă un obiect persistent, ce nu necesită o multitudine de resurse spre a fi întreținut. Clasa care descrie această entitate are ca și corespondent o tabelă dintr-o bază de date relațională, iar fiecare instanță a acestei clase reprezintă o înregistrare din tabelă. Clasa de tip *Entity* se mai numește și POJO-Plain Oriented Java Object*).*
- Crearea depozitelor folosind adnotarea @Repository, acestea conținând specificația metodelor ce vor fi implementate
- Crearea unui @Controller, la nivelul căruia se implementează operațiile dorite
- Modelarea relațiilor între entități (Relațiile pot avea multiplicități precum “1 la 1”, “1 la mai mulți”, “mai mulți la 1” sau “mai mulți la mai mulți”, iar direcția unei relații poate fi unidirecțională sau bidirecțională. Relaționarea între entități e realizată folosind adnotările @OneToOne, @OneToMany, @manyToMany și atributul mappedBy, careprecizează proprietatea din partea referită).
- Înțelegerea și folosirea adnotărilor ce indică legăturile dintre obiecte și date, asigurând persistența acestora (Adnotarea @Id indică o cheie primară, iar @Column realizează maparea unui câmp al tabelei cu o proprietate a clasei. @Transactional asigură faptul că tranzacțiile rulează corespunzător. @Query asigură posibilitatea implementării unei interogări în java, specifică interogărilor SQL).

4.2. Tehnologii necesare în analiza performanțelor SOAP vs REST

4.2.1. Generalități ale conceptului de Serviciu Web

Potrivit sursei [1], „Un Serviciu Web este orice serviciu disponibil pe Internet, care folosește un sistem standardizat de trasmitere de mesaje XML, și nu este legat de niciun sistem de operare sau limbaj de programare.”

Interoperabilitatea Serviciiilor Web este ilustrată și în figura de mai jos, reprezentând două calculatoare ce comunică doar prin mesaje XML.



Figură 4-3 Arhitectura unui Serviciu Web [1]

Pentru a se utiliza transmiterea de mesaje XML există mai multe alternative, precum XML-RPC (Remote Procedure Call) sau protocolul SOAP, dar și metodele GET/POST ale protocolului HTTP, metode care permit parsarea documentelor XML.

Deși nu sunt imperios necesare, e de dorit ca Serviciile Web să aibă următoarele proprietăți:

- *Un Serviciu Web ar trebui să fie auto-descriptibil*

Această proprietate sugerează că publicarea unui nou Serviciu Web ar presupune și publicarea unei interfețe a acelui serviciu. Acesta ar trebui să include o descriere minimă, inteligibilă de către alți programatori ce ar putea integra cu ușurință respectivul Serviciu Web.

- *Un Serviciu Web ar trebui să fie detectabil*

Această proprietate se rezumă la ușurința de a se publica un Serviciu Web, dar și de a se localiza acest serviciu de către părțile interesate.

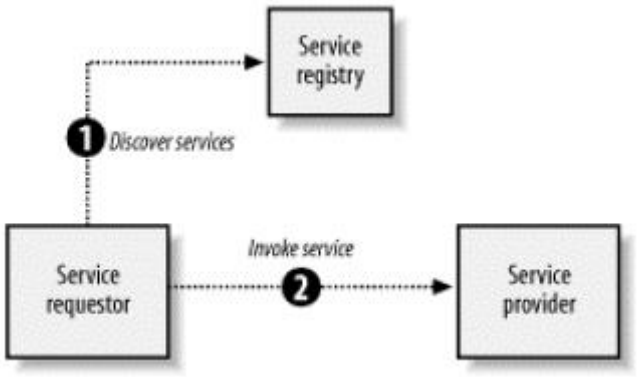
Pentru a sumariza, un Serviciu Web complet este un serviciu care:

- e disponibil prin Internet sau prin rețele private (intranet)
- folosește un sistem standardizat de transmitere de mesaje prin XML
- nu depinde de niciun sistem de operare sau limbaj de programare
- este auto-descriptibil printr-o gramatică XML
- este detectabil (ușor de găsit) prin intermediul unui mecanism simplu

4.2.1.1.Arhitectura unui serviciu Web

Există două modalități de a vizualiza arhitectura unui Serviciu Web: prin examinarea rolurilor individuale ale părților implicate, sau prin studiul stivei de protocoale a serviciului.

4.2.1.2. Rolurile Serviciului Web



Figură 4-4 Rolurile unui Serviciu Web[1]

Există trei roluri esențiale implicate întru Serviciu Web, și anume:

- Furnizorul serviciului (Service Provider)
  -acesta implementează serviciul și îl publica pe Internet
- Clientul serviciului (Service Requestor)
  -reprezintă consumatorul serviciului, care îl utilizează prin deschiderea unei conexiui la rețea și trimitere de cereri bazate pe XML
- Registrul serviciului (Service Registry)
  -este un director centralizat logic al serviciului. Registrul oferă un loc central unde programatorii de servicii pot publica noi servicii sau le pot găsi pe cele existente. [1]

Din figura prezenta mai sus, se observă cum clientul serviciului trimite o cerere către furnizor, invocând astfel Serviciul Web dorit. De asemenea, clientul poate descoperi serviciile prin intermediul registrului de servicii. Clientul va obține datele cerute, care îi vor fi oferite de către furnizor. Totodată, furnizorul a publicat serviciile și acestea se pot regăsi în registrul de servicii.

4.2.1.3. Stiva de protocoale a Serviciilor Web

Aceasta este într-o continua evoluție, dar conșine patru nivele principale, prezentate în cele ce urmează:

- *Service Transport*

Acest nivel este responsabil pentru transmiterea de mesaje între aplicații. Nivelul Transport include protocolul HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol) sau BEEP (Block extensible Exchange Protocol)

- *Transmiterea de mesaje XML*

Nivelul este responsabil pentru codificarea mesajelor într-un format comun XML în așa fel încât mesajele să poate fi înțelese de către toate părțile. Acest nivel include XML-RPC și SOAP.

- *Descrierea de servicii*

Nivelul este responsabil de descrierea interfeței publice a unui Serviciu Web specificat. Acest nivel se remarcă prin folosirea limbajului de descriere a serviciilor web WSDL (Web Service Description Language).

- *Descoperirea serviciului*

Nivelul se ocupă cu centralizarea serviciilor într-un registru comun, precum și de asigurarea ușurinței de a publica sau de a găsi funcționalitatea dorită. UDDI (Universal Description, Discovery and Integration) este elemental essential corespunzător acestui nivel.

Figură 4-5 Stiva de protocoale ale Serviciilor Web

4.2.1.4. Modalităţi de implementare a serviciilor Web

Conceptul de Serviciu Web desemnează o aplicație web de tip client-server. Serverul reprezintă un furnizor de servicii (denumit și "Service Endpoint") și este accesibil unor aplicații client pe baza adresei URL a serviciului. Serviciul Web și clienții care îl folosesc pot fi aplicații ce rulează pe platforme diferite. De asemenea, pot și scrise în limbaje diferite, întrucât comunică prin limbaje standard HTTP, XML, SOAP și așa mai departe.

Proprietatea esențială descrisă astfel pentru Serviciile Web este aceea ca asigură operabilitatea unor aplicații software ce au fost implementate pe platforme diferite, folosindu-se instrumente diferite. Aplicațiile sunt slab-cuplate ("loosely coupled"), ceea ce înseamnă că ele comunică strict prin mesaje standard, neavând alte dependințe.

Furnizorul de servicii expune pe Internet un API (Application Program Interface), acesta reprezentând o serie de metode ce vor putea fi apelate de către clienți. Aplicația client va trebui doar să cunoască URL-ul furnizorului de servicii (a serverului) și metodele prin care poate avea acces la serviciile oferite. Toate aceste elemente vor fi prezentate în conținutul mesajului de tip cerere, care va fi trimis de către client.

Principala diferență care există între o aplicație clasică Web, respectiv un Serviciu Web este ilustrată prin formatul documentelor primite de client și a modului cum sunt folosite: într-o aplicație Web, clientul primește documente HTML transformate de browser în paginile afișate, în timp ce clientul unui Serviciu Web primește un document XML (sau JSON) folosit de aplicația client, dar care nu se afisează direct pe ecran, decât în anumite programe de verificare a Serviciilor Web.

Din punct de vedere al tehnologiilor folosite, se pot distinge două mari categorii de Servicii Web:

- Servicii de tip SOAP (Simple Object Access Protocol)
- Servicii de tip RESTful.

La baza acestora stau două mari arhitecturi, și anume: SOA (Service Oriented Architecture) si REST (REpresentational State Transfer).

4.2.2. Protocolul SOAP

La baza serviciilor de tip SOAP se află paradigma SOC (Service Oriented Computing), care a fost dezvoltată pentru a susține modelul B2B (Business-to-Business). Paradigma SOC se poate defini prin faptul că folosește Serviciile Web ca blocuri pentru construcția și dezvoltarea aplicațiilor distribuite complexe. SOC a evoluat pornind de la alte două paradigme, și anume: OOC (Object-Oriented Computing), respectiv CBC (Component-Based Computing).

Aceste paradigme specifică cele trei entități independente, dar care colaborează între ele: Service Requestors (cei ce invocă serviciile), Service Brokers (cei ce publică) și Service Providers (furnizorii). [27]

Arhitectura SOA (Service Oriented Architecture) este arhitectura ce utilizează paradigma SOC și impune adoptarea unui stil de dezvoltare de aplicații ce sunt considerate servicii, fiind invocate de alte aplicații. SOA presupune faptul că servicii noi se pot crea pornind de la cele existente, însă componentele sistemului trebuie să fie independente una față de cealaltă. Astfel, pe măsură ce apar cerințe diferite, serviciile pot fi recompuse sau se pot organiza diferit.

Principiile de bază ale arhitecturii SOA sunt:

- Serviciile trebuie să partajeze un contract formal;
- Serviciile trebuie să fie slab conectate (*loosely coupled*);
- Serviciile trebuie să ascundă dezvoltatorului detaliile de implementare;
- Serviciile trebuie să ofere suport pentru compunerea cu alte servicii
- (*composability*);
- Serviciile trebuie să poatã fi reutilizate;
- Serviciile trebuie să se execute într-un mod autonom;
- Serviciile nu trebuie să depindã de starea comunicării (*statelessness*), cantitatea
- de informație specifică unei activități ce trebuie reținută fiind minimală;
- Serviciile trebuie să poatã fi ușor de descoperit (*discoverability*).

Prin intermediul SOA, utilizarea și descoperirea serviciilor se pot realiza în mod dinamic, facilitându-se integrarea la nivel de aplicații (A2A - Application to Application) sau de afaceri (B2B - Business to Business). Suplimentar, prin intermediul serviciilor se poate realiza managementul proceselor de afaceri (BPM – Business Process Management).

Metodologia de proiectare și dezvoltare a aplicațiilor aliniate prevederilor SOA poartă numele de SOAD (Service-Oriented Analysis and Design). Una dintre propunerile de astfel de metodologii este cea publicată de IBM în 2004 - SOMA (Service-Oriented Modelling and Architecture).

4.2.3. SOAP, WSDL și UDDI

Pentru invocarea, definirea și regăsirea serviciilor Web, sunt necerare următoarele componente: protocolul SOAP, WSDL-ul și UDDI-ul. [27]

Pentru a se ușura transmiterea mesajelor între mașini diferite, permițând interacțiuni complexe și având conținut oricât de complex, a fost necesară dezvoltarea unui protocol de comunicare (protocol de transport). Acesta trebuie să asigure support pentru asigurarea extensibilității, în același timp ținându-se cont de problemele de securitate, fiabilitate și performanță care pot apărea. Pentru a se transmite mesajele în formal XML, dar și pentru primirea răspunsului serviciului Web invocat, e necesară folosirea unui limbaj, care să fie comun mașinilor ce comunică. Acesta definește, practic, protocolul de comunicare.

Protocolul SOAP (Simple Object Access Protocol) este apărut ulterior protocolului XML-RPC, care permitea apeluri și procedure la distanță, fiind proiectat pentru a fi cât mai simplu. SOAP este astăzi foarte cunoscut și utilizat, înlocuind protocoale mai vechi, precum RMI, CORBA; DCOM, JAX-RPC sau altele.

Conform protocolului SOAP, între clientul care invocă un serviciu și furnizorul serviciilor se schimbă mesaje SOAP, care sunt înțelese de ambele părți, indiferent de modul de implementare a fiecăreia.

Mesajul SOAP este, practic, un document de tip XML ce respectă o anumită structură, având trei părți esențiale: un plic (envelope), un set de reguli de codificare (encoding rules) și o convenție de reprezentare (reprezentation).

Mai exact, elementele unui mesaj de tip SOAP sunt:

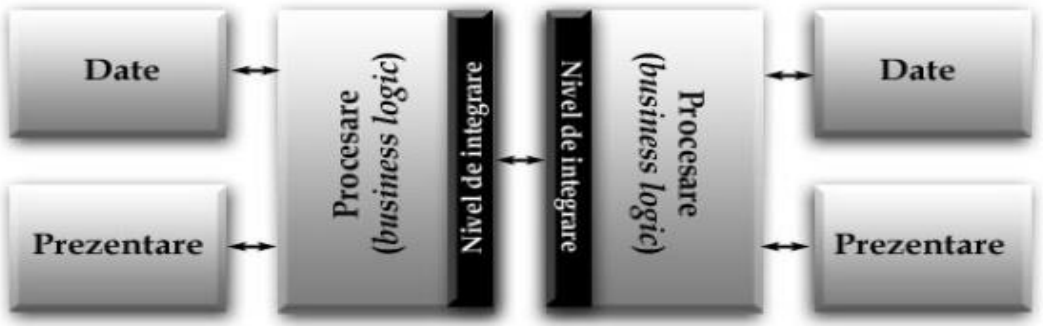
- **Envelope**(plicul) - e un element obligatoriu care identifică documentul XML ca un mesaj SOAP; De asemenea, este elemental rădăcină al mesajului SOAP și acționează ca și un container pentru informația ce trebuie livrată
- **Header** (optional)
  - poate conține informații generale despre mesajul SOAP
  - daca este prezent, trebuie să fie primul copil al elementului Envelope
  - fiecare bloc antet are asociat un spațiu de nume calificat, în acord cu regulile din SOAP Schema)
- **Body** (obligatoriu) – conține apelul sau răspunsul mesajului
- **Fault** (optional)
  - oferă informații despre erori care apar în timpul procesării mesajului SOAP;
  - poate apărea o singură dată în conținutul mesajul
  - conține, obligatoriu, două elemente-copil: *Code* și *Reason*.

Serviciile Web de tip SOAP sunt ușor de înțeles, întrucât informațiile cu privire la cerințele funcționale sunt expuse utilizatorului, prin intermediul documentelor de tip WSDL (Web service Description Language). Cu alte cuvinte, acesta reprezintă un Limbaj de Descriere a Serviciilor Web, fiind cel mai cunoscut limbaj de descriere a Serviciilor Web. Această descriere se referă la specificarea locației și descrierea operațiilor(metodelor) permise.

Ca un scurt istoric, specificația WSDL initial a luat naștere în anul 2000, prin fuziunea a două limbaje de descriere a serviciilor: NASSL (Network Application Service Specification Language) propus de IBM și

SDL (Service Description Language) provenind din partea Microsoft.Versiunea WSDL 1.1 a fost publicată în 2001, fiind foarte populară și utilizată. Mai apoi, a evoluat spre WSDL 1.2, iar în prezent se lucrează la versiunea 2.0 a specificației WSDL.

Conceptual, documentul WSDL reprezintă un "contract" între un mesaj de tip cerere și unul de tip răspuns și poate fi comparat cu interfețele în Java, ce sunt, de asemenea, văzute ca și contracte. O diferență esențială este că documentul WSDL poate fi privit și ca o platformă, dar și ca un limbaj de marcare, folosit în principal pentru a descriere serviciile de tip SOAP.



Figură 4-6 Prezentarea conceptului de WSDL

Descrierea unui serviciu Web prezintă două părți principale: definiția abstractă și definiția concretă.

Partea abstractă conține descrierea serviciului în termeni de mesaje trimise și recepționate, sub controlul unui sistem de tipuri. Modelul schimbului de mesaje definește succesiunea și numărul mesajelor. O operație asociază modelul schimbului de mesaje cu unul sau mai multe mesaje, iar o interfață grupează aceste operații independent de transport sau de metoda de serializare.

Partea concretă a descrierii prezintă legăturile (*bindings*) între modul de transport și formatul de serializare pentru interfețe. Elementul numit *endpoint* face legătura cu o adresă, iar elementul *service* grupează toate endpoint-urile care implementează o interfață comună.

Aceste elemente sunt descrise în tabelul de mai jos:

Tabel 4-1 Modelul conceptual WSDL

Interfața serviciului (definiție abstractă)	Mesaje (*messages*)
	Operații (*operation*)
	Interfață *(interfaces*)
Implementarea serviciului (definiție concretă)	Legare (*bindings*)
	Serviciu (*service*)
	Punct terminal (*endpoint*)

Pentru a descrie Serviciile Web, WSDL furnizează un set de componente și proprietăți ale acestora:

- **Definitions** (elemental rădăcină, care poate fi văzut ca un container ce conține toate informațiile și atributele associate unui Serviciu Web)
- Atributul *targetNamespace* e obligatoriu, de tipul *anyURI*.
- **Messages** (element-copil direct al marcatorului *definitions*, având rol în reprezentarea mesajelor intrare/ieșire)
- **Interface** (include un set de operații abstracte și poate extinde una sau mai multe alte interfețe)
- **Types** (container pt definirea tipurilor de date)
- **Message** (o definiție abstractă a date ce va fi comunicată)
- **Operation** (o definiție abstractă a unei acțiuni suportată de serviciu, ce constă dintr-un grup de mesaje de intrare/ieșire legate între ele)
- **Port Type** (un set abstract de operații suportate de unul sau mai mai multe "endpoint-uri")
- **Binding** (un protocol concret și specificațiile unui format de date pentru un "port type")
- **Port** (un singur endpoint definit ca o combinație de "binding" și "network address"
- **Service** (o colecție de "endpoint-uri" legate, la nivelul cărora Serviciul Web poate fi accesat)

4.2.3.1. UDDI [12]

O altă cerință pe care o respectă Serviciile Web este posibilitatea ca o anumită funcționalitate să poată fi regăsită prin intermediul unor interogări ce au în vedere regăsirea informațiilor dorite. Catalogul UDDI (Universal Description, Discovery, and Integration) oferă o bază de date distribuită a Serviciilor Web din prisma tipurilor de afaceri electronice pe care acestea le pot modela. UDDI furnizează un registru (Service

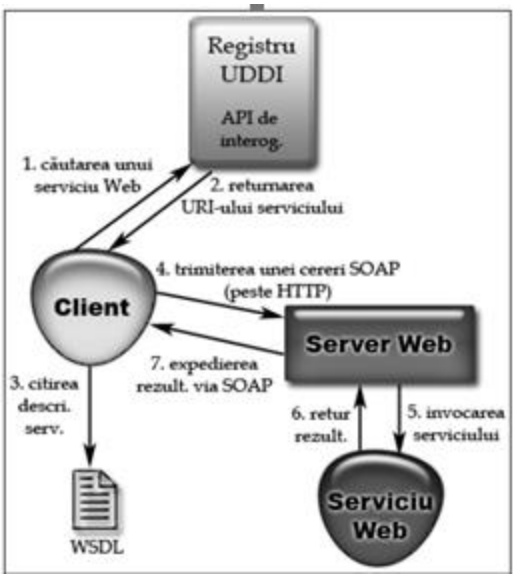
Registry) pentru a localiza Serviciile Web. Acesta poate fi văzut ca un Domain Name Service (DNS) pentru aplicațiile business. UDDI permite:

- **Service Publication** – UDDI definește operațiile ce permit organizațiilor să distribuie serviciile web
- **Query** – UDDI definește operații care permit utilizatorilor să extragă informații despre un serviciu folosind registru UDDI
- **Classification** – UDDI definește operații care permit clasificarea business-ului și al serviciului în acord cu standardul taxonomiei.

UDDI constă in 3 elemente numite pagini:

- **White pages (paginile albe)** – conțin un contact de bază pentru fiecare furnizor de servicii (Service Provider), cu alte cuvinte prezintă informații generale referitoare la o companie ce furnizează servicii
- **Yellow pages (paginile aurii)** – conțin mai multe detalii despre companie, includ alte beneficii oferite de companie, include scheme de clasificare a companiilor sau a serviciilor disponibile

  Green pages (paginile verzi) – conțin informațiile legate de Serviciile Web (interfețe, locații URL, date necesare pentru a localiza și utiliza/invoca Serviciile Web) registrul UDDI interogat prin SOAP pentru a se obține descrierea WSDL



Figură 4-7 Relațiile dintre un client, un serviciu Web și [12]

Functionalitățile de căutare sunt implementate prin servicii Web, astfel încât accesul la registrul UDDI se realizează folosind protocolul SOAP. UDDI pune la dispozitie documentul WSDL corespunzător unui serviciu compatibil cu cererea formulată.

Pașii necesari pentru a utiliza un registru UDDI în contextul arhitecturii SOA:

1. Un Web Service Provider publică informații despre un serviciu web (luate din fișierul WSDL) în registrul UDDI

2. Un Web Service Requester caută în UDDI pentru a găsi serviciul care se potrivește nevoilor sale
3. UDDI-ul trimite descrierea serviciului la Web Service Requester
4. Web Service Requester-ul se conectează la furnizorul de Servicii Web utilizând protocolul SOAP pentru a obține fișierul WSDL care descrie funcționalitatea serviciului web.
5. Operațiile Serviciului Web sunt invocate prin intermediul SOA

4.2.4. Arhitectura REST[12]

REST (Representational State Transfer) este o arhitectură prin care se dezvoltă aplicații Web. Filosifia REST presupune că rezultatul unei procesări va duce la returnarea unei reprezentări a unei resurse Web. Orice accesare a unei reprezentări de către aplicația client va duce la trecerea acesteia într-o stare ce va fi schimbată în urma unui transfer de date, adică accesarea altei reprezentări, pe baza traversării unei legături hypertext, desemnată de un URI (Uniform Resource Locator), care este inclusă în reprezentarea resursei inițiale.

Mesajele se vor transmite între client și server, însă starea comunicării acestora nu va fi reținută, iar această proprietate specifică arhitecturii REST poartă denumirea englezească de **stateless**. Astfel, conexiunile fiind stateless, fiecare cerere este independent, neținându-se cont de context.

Datele între părțile ce comunică se transferă folosind protocolul HTTP, ele fiind reprezentate în format XML (sau în alt format). Adresabilitatea se rezolvă prin URI, spatial Web putând fi astfel considerat drept un sistem REST.

Resursele Web se pot accesa printr-o interfață generic pusă la dispoziție de metodele protocolului HTTP, acestea fiind: GET, POST, PUT și DELETE. Cele mai utilizate în momentul actual sunt metodele GET și POST.

REST se consideră a fi o viziune complementară de implementare și utilizare a serviciilor Web, în locul mesajelor SOAP (al căror format e considerat de unii dezvoltatori prea complex în anumite situații) recurgându-se la reprezentări XML mai simple, care se numesc POX (Plain Old XML).

O aplicație Web dezvoltată conform principiilor REST prezintă o arhitectură diferită de una în stilul RPC (acesta din urmă fiind adoptat și de protocolul SOAP). În cazul RPC, punctul focal e reprezentat de mulțimea de operații ce pot fi executate (numite și verbe), pe când în viziunea REST totul se axează pe diversitatea resurselor disponibile (denumite și substantive).

În momentul de față, numeroase organizații folosesc arhitectura REST pentru a furniza servicii Web clienților. Printre cele mai cunocute se numără Amazon, Bloglines, del.icio.us, eBay, Google și Yahoo!. De asemenea, există numeroase cadre ce se pot folosi pentru dezvoltarea de aplicații Web în stil REST, și anume: JAX-WS (Java API for XML: Web Services), Tonic (pentru PHP), Ruby on Rails, Zope (pentru Python) și așa mai departe.

4.2.4.1. Principii arhitecturale

Proprietățile arhitecturale ale stilului architectural REST sunt:

- Performanța: interacțiunile dintre componente și eficiența rețelei pot fi factori dominanți în performanța percepută de utilizator
- Scalabilitatea: pentru a suporta un număr mare de componente și interacțiuni între componente
- Simplitatea interfețelor
- Portabilitatea componentelor

4.2.4.2. Contrângeri arhitecturale

Constrângerile arhitecturii REST sunt:

Stateless: fiecare request de la client conține toate informațiile necesare spre a servi acest request, iar starea de sesiune e păstrată în client. Sesiunea poate fi transferată de către server unui alt serviciu (de exemplu, în cazul unei baze de date, pentru a persista starea pe o anumită perioadă de timp și pentru a permite autentificarea . Clientul începe să trimită cereri (request-uri) în momentul în care este pregătit să facă o tranziție la o nouă stare. Cât timp unul sau mai multe request-uri nu sunt finalizate, clientul se consideră a fi într-o stare de tranziție. Reprezentarea fiecărei stări a aplicației conține link-uri ce pot fi utilizate ulterior atunci când clientul decide să inițieze o nouă tranziție a stărilor.

Client-Server: serverul este separat de clienți prin intermediul unei interfețe uniforme. Clienții nu sunt interesați de detalii privind baza de date, de modul cum sunt stocate datele, această atribuție revenindu-i serverului, astfel îmbunătățindu-se portabilitatea codului client. Totodată, acest lucru duce la simplitate și scalablitatea serverelor.

Cachable: În mediul web, clienții și intermediarii pot să "cache-uiască" răspunsurile. Un bun management al caching-ului reduce sau elimină complet o parte din interacțiunile client-server, îmbunătățind astfel scalabilitatea și performanța.

Sistem bazat pe layere: Clientul nu știe dacă este conectat direct la server sau dacă există un intermediar până la server. Serverele intermediare pot îmbunătăți scalabilitatea prin implementarea unui „load balancer" și prin punerea la dispoziție a resurselor cached comune.

Interfața uniformă: Aceasta reprezintă constrângerea fundamentală în cazul dezvoltării oricărui serviciu RESTful. Interfața uniformă simplifică și decuplează arhitectura, lucru ce permite ca fiecare parte să evolueze separat.

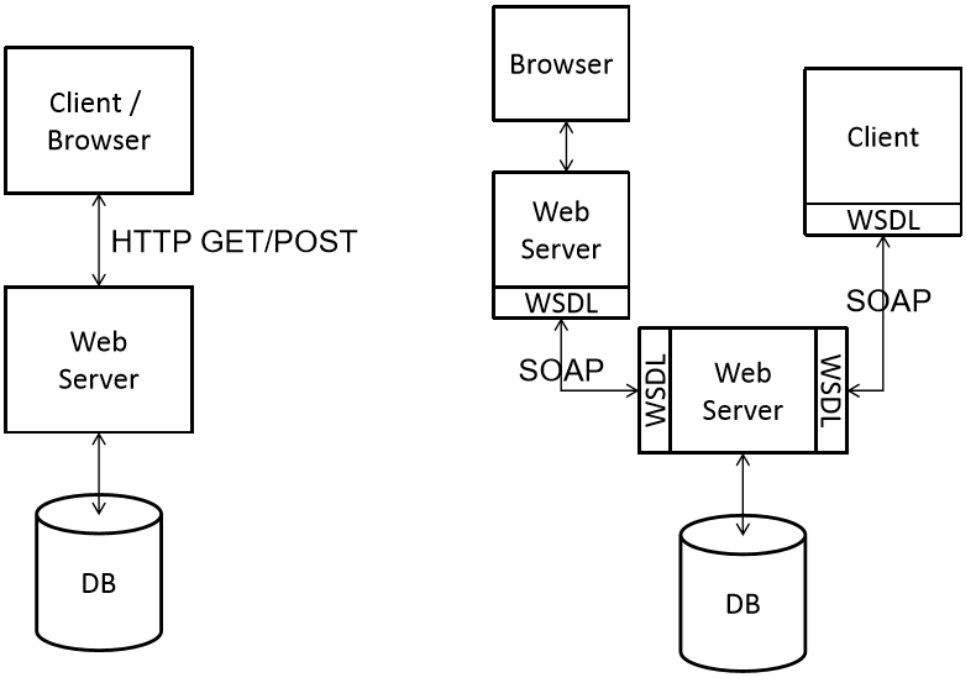
4.2.4.3. Formatul JSON [12]

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, prin procesul 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, fiind utilizat alături de acest limbaj. Tipul de media pe care trebuie să îl transmită un document JSON este *application/json*. Extensia fișierelor JSON este *json*.

Un avantaj important al formatului JSON în comparație cu XML este viteza. Fiind un format mai simplu, analizoarele JSON sunt mai simplu de construit și evident mai rapide. Viteza este deosebit de importantă în aplicațiile AJAX, iar deoarece JSON se pretează foarte bine pentru interschimbarea de date prin XMLHttpRequest, acest format devine din ce în ce mai popular în dezvoltarea aplicațiilor Web 2.0.

Fiind prezentate anterior conceptele de *serviciu web,* precum și detalii legate de implementarea acestora împreună cu tehnologiile folosite, se prezintă un prim element de evaluare comparativă, prin arhitecturile conceptuale ale serviciilor REST și SOAP.

Astfel, în figura următoare este prezentat modul de funcționare al fiecărei arhitecturi. În cazul arhitecturii REST, clientul invocă serviciul web prin trimiterea unei cereri de tip GET sau POST, urmând să primească un răspuns în urma procesării cererii. În ceea ce privește cea de-a doua diagramă, mesajele de tip SOAP implică și WSDL-ul, mesajele fiind parsate în format XML.Se poate observa, astfel, că mesajele de tip SOAP sunt mai complexe decât cele REST.



Figură 4-8 Arhitectura REST vs SOAP

4.2.5. SoapUI

SoapUI [10] este cel mai cunoscut instrument pentru simularea unui client de servicii Web (prin afișarea cererilor client și a răspunsurilor primite), pentru testarea si dezvoltarea de servicii SOAP si RESTful. Cu alte cuvinte, SoapUI este utilitarul de top în testarea serviciilor web.

SoapUI reprezintă o aplicație desktop gratuită și “open source” folosită pentru inspectarea, invocarea și dezvoltarea serviciilor web, pentru simularea și mocking-ul serviciilor, dar și pentru testarea funcțională, la încărcare și de integrarare a serviciilor web (atât RESTful, cât și SOAP).

SoapUI asigură suport atât pentru testarea serviciilor implementate cu SOAP, cât și pentru cele implementate cu REST. Astfel, la crearea proiectului exista două opțiuni diferite (Crearea unui proiect SOAP sau a unuia REST). Testele sunt organizate în proiecte, astfel că fiecare proiect poate avea mai multe suite de test, iar fiecare suită poate avea mai multe cazuri de test (Test cases). Acestea sunt create din mai mulți pași de test (test steps). Printr-un astfel de pas de test se organizează logica de validare a unei anumite funcționalități dintr-un serviciu web. O tehnică recomandată în crearea testelor funcționale este metoda cascadă, care validează operațiile pe rând, în ordinea în care acestea au fost create și executate. De asemenea, instrumentul oferă multiple validări: se validează conținutul răspunsului, sau dacă data întoarsă trebuie să respecte un anumit timp, dacă se trimite un cod de eroare, precum și mesajul afișat în acest caz.

4.2.6. Apache Groovy

Apache Groovy [10] reprezintă un limbaj de programare puternic, optional tipizat și dynamic, ce deține capabilități statice de compilare pentru platforma Java, menită să îmbunătățească productivitatea dezvoltatorilor datorită unei sintaxe concise, familiare și ușor de îmvățat. Groovy se integrează fără probleme cu orice program Java și oferă imediat aplicației funcții puternice, capabilități de scripting, autorizarea limbajului specific domeniului, meta-programarea și programarea funcțională de execuție și de compilare.

SoapUI oferă opțiuni extinse de scripting, folosind fie Groovy, fie Javascript (de la SoapUI 3.0) ca limbaj de scripting, fiind selectat la nivel de proiect limbajul ce urmează a fi folosit. Groovy simplifică foarte mult scriptingul API-urilor, motiv pentru care este preponderent utilizat.

Scripturile pot fi utilizate în următoarele locuri în SoapUI:

- Ca parte a unui TestCase cu Groovy Script TestStep, permițând testelor să efectueze practic orice funcționalitate dorită
- Înainte și după rularea unui TestCase sau a unui TestSuite pentru inițializare și curățare înainte sau după executarea testelor.
- La pornirea sau oprirea unui MockService pentru a inițializa sau a curăța starea MockService
- La deschiderea / închiderea unui proiect, pentru inițializarea sau curățarea setărilor legate de proiect
- Ca o sursă de date dinamică sau de date de date, cu pașii de testare corespunzători datelor DataSource / DataSink
- Pentru furnizarea dispecerizării dinamice MockOperation.
- Pentru crearea conținutului dinamic MockResponse
- Pentru a crea afirmații arbitrare cu Script Assertion
- Pentru a extinde însăși SoapUI (a se vedea Extinderea SoapUI), pentru a adăuga funcționalitate arbitrară la nucleul SoapUI

Toate scripturile au acces la un număr de variabile specifice situației, întotdeauna incluzând un obiect de jurnal pentru logarea în Log Groovy și un obiect context pentru a efectua PropertyExpansions specific pentru context sau manipulare proprietate, dacă este cazul. O variabilă specifică contextului este întotdeauna disponibilă pentru accesarea directă a modelului de obiect SoapUI.

Capitolul 5. Proiectare de Detaliu si Implementare

În acest capitol se vor prezenta detaliile legate de implementarea serviciilor web, altfel spus se va prezenta modul în care s-a creat contextul necesar realizării testelor de performanță a serviciilor web. Capitolul conține detalii legate de arhitectura sistemului, componentele implicate, precum și modul în care acestea sunt interconectate între ele.

5.1. Arhitectura sistemului

Sistemul dezvoltat cu scopul de a furniza același set de operații atât în REST, cât și SOAP, a fost realizat în limbajul de programare Java EE, folosindu-se ca mediul de integrare Eclipse Jee Neon. Acesta oferă dezvoltatorilor un set de unelte ce ajută în scrierea codului, respectiv implementarea serviciilor web.

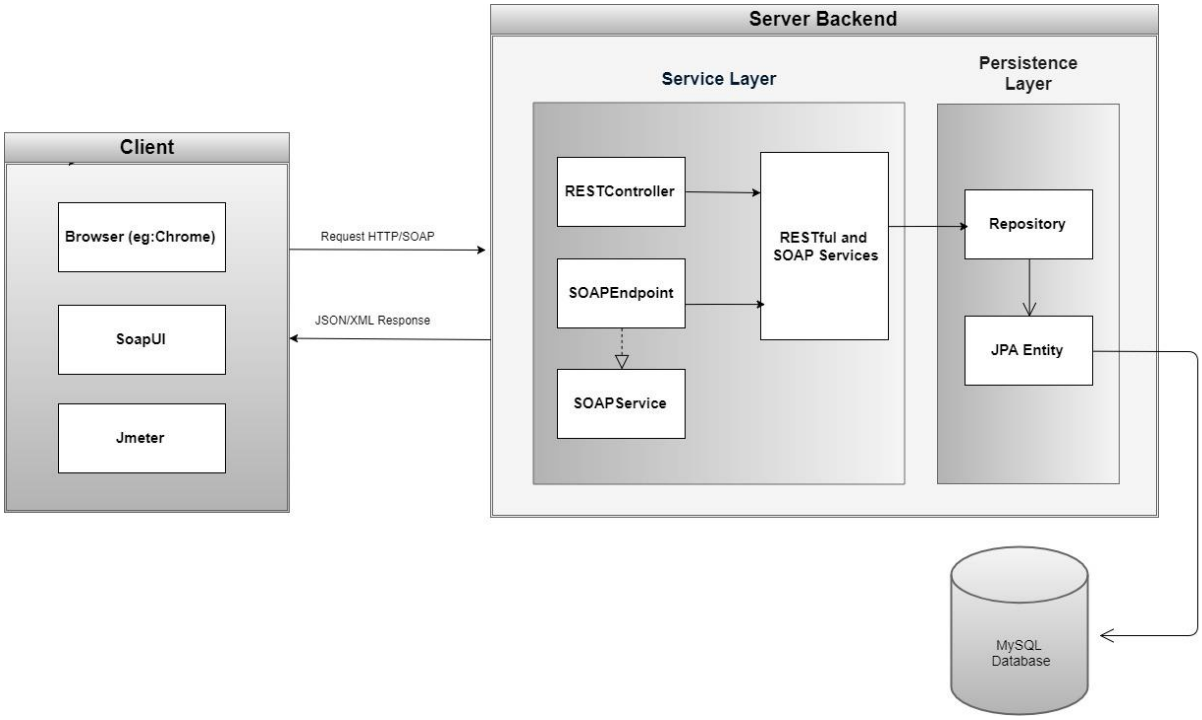
Arhitectura utilizată de sistem este cea client-server, pe baza căreia clientul trimite anumite request-uri către server, iar acesta din urmă procesează mesajul primit, trimițând mai apoi un răspuns către client. Arhitectura client-server se bazează, astfel, pe existența unei aplicații distribuite ce partajează sarcinile între furnizorii de servicii (servere) și cei care solicită servicii (clienți). Comunicarea între clienți si servere se realizează, de cele mai multe ori, prin intermediul Internetului, precum în sistemul de față în cazul serviciilor RESTful. De asemenea, clientul și serverul partajează mesaje prin utilizarea unui format comun de date, precum formatul XML în cazul mesajelor SOAP.

În cele ce urmează, se va prezenta arhitectura conceptuală a sistemului, aceasta fiind ilustrată în figura 5.1. După cum se poate observa, clientul este reprezentat fie de browser, fie de SoapUI sau JMeter (acestea din urmă fiind instrumente speciale utilizate în testarea serviciilor web prin simularea trimiterii de request-uri).

Partea de server reprezintă o aplicație implementată folosind SpringBoot, iar arhitectura acesteia este pe nivele, astfel ilustrându-se ***Service Layer***, respectiv ***Persistance Layer***.

Nivelul ***Service*** conține serviciile web implementate. Ele sunt descrise prin endpoint-uri separate pentru RESTful și SOAP, însă implementarea lor este una comună.

Nivelul ***Persistance*** se ocupă de gestionarea datelor aflate în baza de date MySQL. Persistența datelor se realizează prin folosirea framework-ului *Spring Data,* ce extinde JPA (Java Persistence API).



Figură 5-1 Arhitectura conceptuală

5.1.1. Rolul componentelor sistemului

În cele ce urmează se vor prezenta, pe scurt, componentele implicate și rolul acestora, pentru a se înțelege mai bine atât importanța acestora cât și modul în care au fost concepute. Întrucât implementarea efectivă a serviciilor s-a realizat la nivel de server, această componentă are o pondere mai mare, motiv pentru care i se vor prezenta mai amănunțit componentele fiecărui nivel.

5.1.1.1. Service Layer

Acest nivel este responsabil de interceptarea request-urilor primite de la client, precum și de construirea și trimiterea răspunsurilor înapoi către client. Atât serviciile RESTful, cât și cele SOAP sunt implementate la acest nivel.

În ceea ce privește serviciile RESTful, ele pot fi invocate de către client prin trimiterea unui request la adresa *http://localhost:8080/{controller}/{method}*. Spre exemplu, dacă se dorește listarea tuturor studenților, request-ul trimis va *http://localhost:8080/students/all-students*, sau dacă se dorește listarea studenților de la o anumită specializarea, request-ul va fi *http://localhost:8080/students/bySpecializare*. Clasa care se ocupă cu implementarea acestor operații are funcție de controller, fiind adnotată cu *@RestController*. În funcție de metodele HTTP care vor putea fi accesate la un anumit URL, metodele implementate în Java pot avea adnotări precum *@RequestMapping* (permite request-uri de tip POST,PUT,GET și DELETE), *@GetMapping* (permite doar request-uri de tip GET), *@PutMapping* (permite doar request-uri de tip PUT) și așa mai departe.

Spre exemplu, metoda folosită pentru a modifica media unui student este prezentată mai jos. Această metodă acceptă doar request-uri de tip PUT, având adnotarea *@PutMapping*.

```
@PutMapping("/student/update")
public String updateStudent(@RequestParam Double media, @RequestParam Long id)
{
studentService.updateStudent(media, id);
return "Media studentului a fost modificata!";
}
```

După cum se poate observa în fragmentul de cod de mai sus, parametrii metodei sunt media și id-ul studentului, parametri ce vor fi cunoscuți de către clienți în scopul trimiterii unui request.

În cazul serviciilor de tip SOAP, metodele sunt prezentate în cadrul unei clase având adnotările *@Component* și *@WebService*, precum mai jos:

```
@Component
@WebService(endpointInterface
"com.licenta.ws.soap.wsService.StudentWSService")
public class StudentSOAPEndPoint implements StudentWSService{
@Autowired
private StudentService studentService;
@Override
public String saveStudent(String nrMatricol,String nume,String cnp,String
specializare,String adresa,Double media) {
Student s=new Student(nrMatricol, nume,cnp, specializare, adresa, media);
studentService.save(s);
String sir="Studentul cu numarul matricol " +nrMatricol+ " a fost salvat! ";
return sir;
}
}
```

Această clasă implementează o interfață având tot adnotarea *@WebService*, interfață ce definește toate metodele ce se vor implementa, precum și url-ul de unde vor fi accesate.

```
@WebService
public interface StudentWSService {
@WebMethod(operationName = "saveStudent",
action = "https://localhost/wsdl/student/saveStudent")
String saveStudent(String nrMatricol,String nume,String cnp,
        String specializare,String adresa,Double media);
}
```

Aceste metode se vor regăsi în cadrul wsdl-ului care va putea fi accesat la adresa prezentată mai jos: *http://localhost:8081/wsdl/student?wsdl* .

Precum se observă atât din diagrama conceptuală, cât și din codul expus anterior, serviciile RESTful și SOAP sunt implementate identic, fiind apelate din locuri separate (din RESTEndpoint, respectiv din SOAPService).

Astfel, se prezintă clasa StudentService, adnotată cu @Component, la nivelul căreia sunt implementate operațiile specificate de RESTController, respectiv de SOAPWebService.

```
@Component
public class StudentService {

	@Autowired
	StudentRepository studentRepository;
	@Autowired
	LicentaRepository licentaRepository;

	//OK
	public void save(Student student){
		studentRepository.save(student);
	}
}
```

Fiecare entitate a bazei de date are o clasă Service asociată, în interiorul căreia sunt prezentate operațiile ce vor fi implementate. Acestea sunt: StudentService, ProfesorService, LicentaService.

În clasa StudentService s-au implementat operațiile:

- *Save(Student student) : void* – această metodă are rolul de a adăuga un nou student în listă
- *getStudentByNrMatricol (String nrMatricol) : Student* – această metodă returnează un student din listă, pe baza numărului său matricol
- *findAll() : List<Student>* – această metodă returnează lista completă de studenți
- *delete(Long id) : void* – această metodă are rolul de a șterge un student din listă
- *getStudentsBySpecializare (String specializare) :* List<*Student*> – această metodă returnează lista completă de studenți de la o anumită specializare
- *getStudentByGrades(double medie1, double medie2) : List<Student>* – această metodă returnează lista studenților cu medii cuprinse în intervalul dat
- *findById(Long id) : Student* – această metodă returnează un student găsit după id-ul său
- *deleteByNrMatricol(String nrMatricol) : void* – această metodă are rolul de a șterge un student din listă
- *deleteAndRetrieveStudents(String nrMatricol) : List<Student>* – această metodă șterge un student din listă, apoi returnează noua listă de studenți
- *updateStudent(Double media, Long id) : void* – această metodă are rolul de a modifica media unui student având un anumit id
- În clasa ProfesorService s-au implementat operațiile:

- *Save(Profesor prof) : void* – această metodă are rolul de a adăuga un nou profesor în listă
- *findAll() : List<Profesor>* – această metodă returnează lista completă de profesori
- *delete(Long id) : void* – această metodă are rolul de a șterge un profesor din listă
- *getTeachersByDepartament(String departament) :* List<*Profesor*> – această metodă returnează lista completă de profesori de pe un anumit departament
- *getStudentFromSpecificProfesor() : List<Student>* – această metodă returnează lista studenților asignați unui anumit profesor
- *getStudentByComisie(String comisie) : List<Student>* – această metodă returnează lista studenților asignați profesorilor dintr-o anumită comisie
- În clasa LicentaService s-au implementat operațiile:
- *findAll() : List<Licenta>* – această metodă returnează lista completă de licențe
- *getLicentaForStudent(Long studentId) : void* – această metodă returnează licența unui student cu id-ul dat
- *deleteLicentaForStudent (Long studentId) : void* – această metodă are rolul de a șterge o licență a unui student cu id-ul dat
- *getThesisByComisie(String comisie) : List<Licenta>* – această metodă returnează lista licențelor ce corespun profesorilor dintr-o anumită comisie
- *getThesisByDepartament(String departament) : List<Licenta>* – această metodă returnează lista licențelor ce corespun profesorilor dintr-un anumit departament

5.1.1.2. Persistence Layer

Acest nivel este responsabil de gestionarea datelor preluate din baza de date MySQL. Datele sunt gestionate ca entități JPA, prin intermediul claselor din pachetul *com.licenta.model*, astfel încât fiecare tabelă din baza de date MySQL are o clasă corespondentă în Java, numită entitate. Aceste clase vor fi adnotate cu @Entity, cuprinzând și numele entității (al tabelei). De asemenea, atributele clasei vor conține adnotări precum @Column pentru numele câmpurilor tabelelor, sau, spre exemplu @OneToOne, @ManyToOne pentru a indica relațiile dintre tabele.

Astfel, pentru tabela denumită Student și având atributele *id*, *nume* și *nrMatricol*, va exista clasa Student descrisă mai jos:

```
@Entity(name="student")
public class Student {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    @Column(name="nrMatricol")
    private String nrMatricol;
    @Column(name="nume")
    private String nume;  }
```

Entitățile prezente în proiect sunt: *Student, Profesor, Licență* și *ProfStud.*

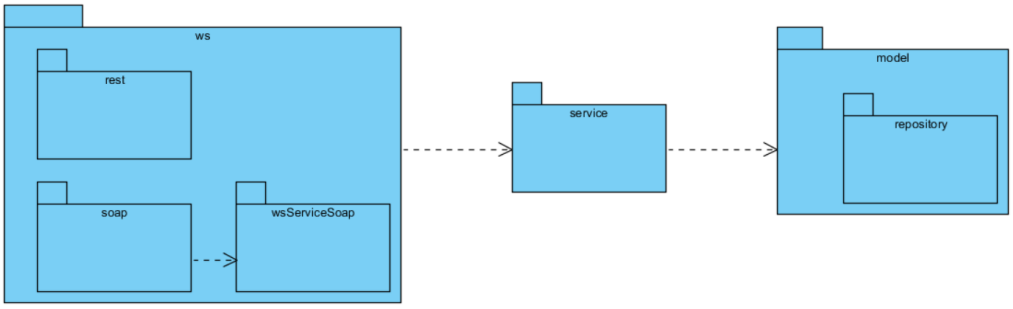
Entitățile prezentate sunt folosite în pachetul *Repository,* acesta conținând interfețele ce descriu metodele ce urmează a fi implementate prin serviciile web. Acestea sunt: *StudentRepository, ProfesorRepository, LicențăRepository* și *ProfStudRepository.*

Fiecare dintre acestea extinde interfața CrudRepository, implementând metodele necesare în vederea construirii serviciilor.

5.2. Structura pachetelor și a claselor sistemului

După prezentarea arhitecturii conceptuale a sistemului e necesară prezentarea pachetelor și a claselor, pentru o mai bună înțelegere a modului în care au fost implementate operațiile.

Aplicația conține următoarele pachete: *com.licenta.model, com.licenta.model.repository, com.licenta.service, com.licenta.ws.rest, com.licenta.ws.soap, com.licenta.ws.soap.wsService,* iar structura acestora este prezentată



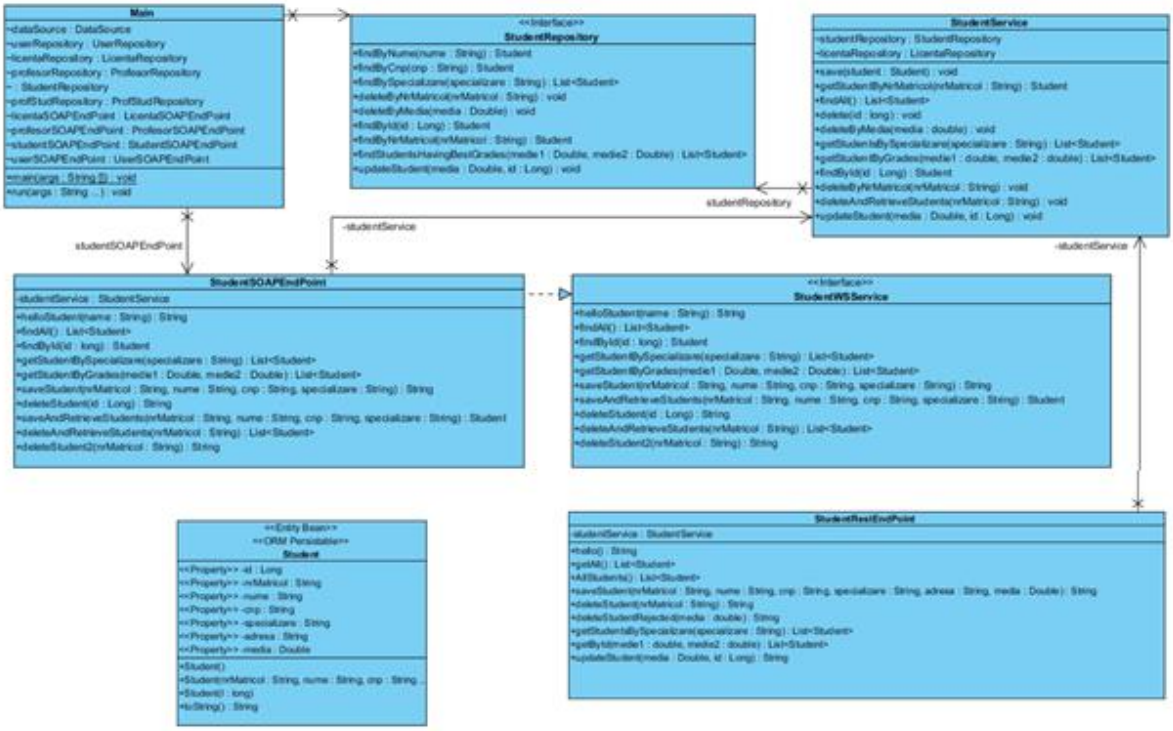
Figură 5-2 Diagrama de pachete a aplicației

Pentru a se clarifica fluxul de transmitere a mesajelor și a modului în care au fost implementate serviciile RESTful și SOAP, dar și pentru a se detalia structura pachetelor prezentate anterior, se prezintă în continuarea diagrama de clase.

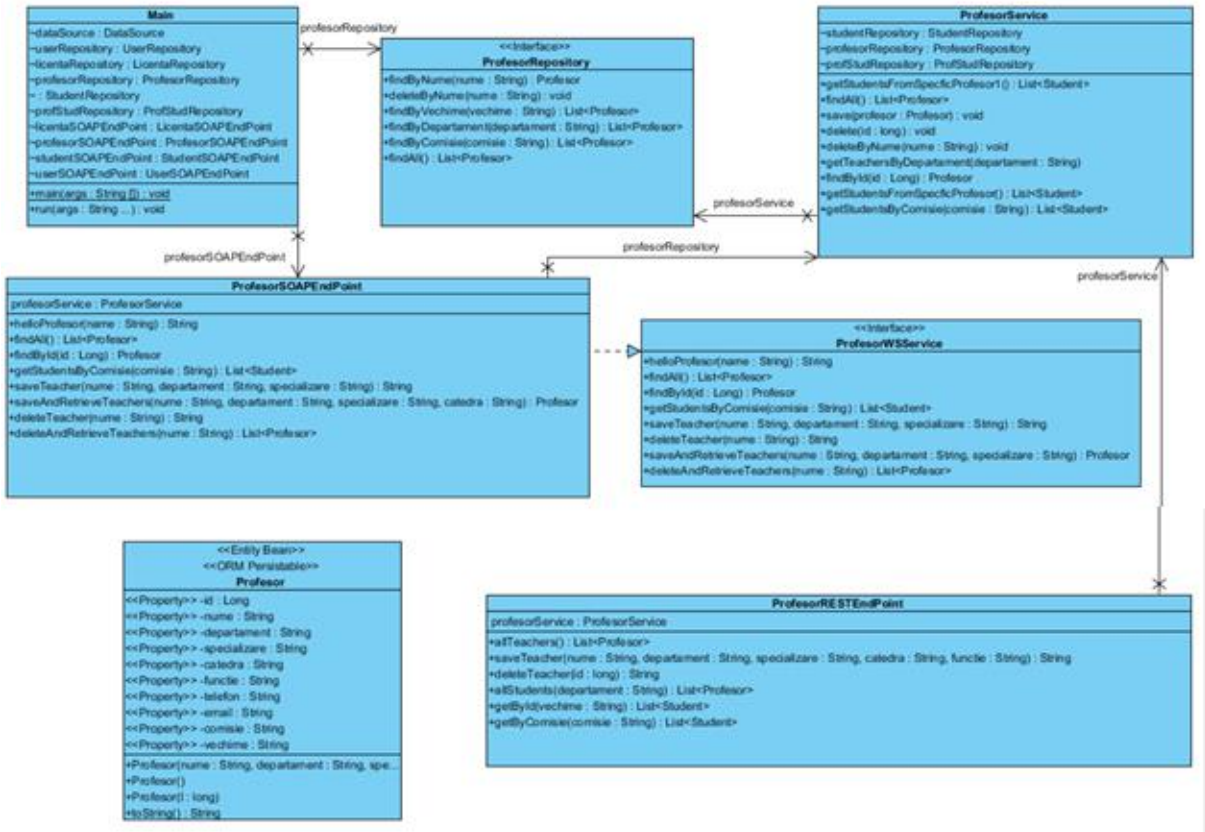
Pachetul com.licenta.model conține entitățile ce reprezintă tabelele existente în baza de date, iar metodele de acces la date sunt definite în cadrul interfețelor prezente în pachetul com.licenta.model.repository. Aceste metode urmeaza sa fie implementate in cadrul claselor din pachetul com.licenta.service. Mai apoi, metodele necesare vor fi apelate atât din pachetele com.licenta.ws.rest, cât și din com.licenta.ws.soap, astfel încât se remarcă ideea că serviciile RESTful, respectiv serviciile de tip SOAP au la bază implementarea acelorași operații.

În figura 5-3 sunt descrise clasele necesare implementării operațiilor pentru

entitatea *Student,* iar în figura 5.-4 sunt prezentate operațiile realizate pentru entitatea de tip *Profesor*.

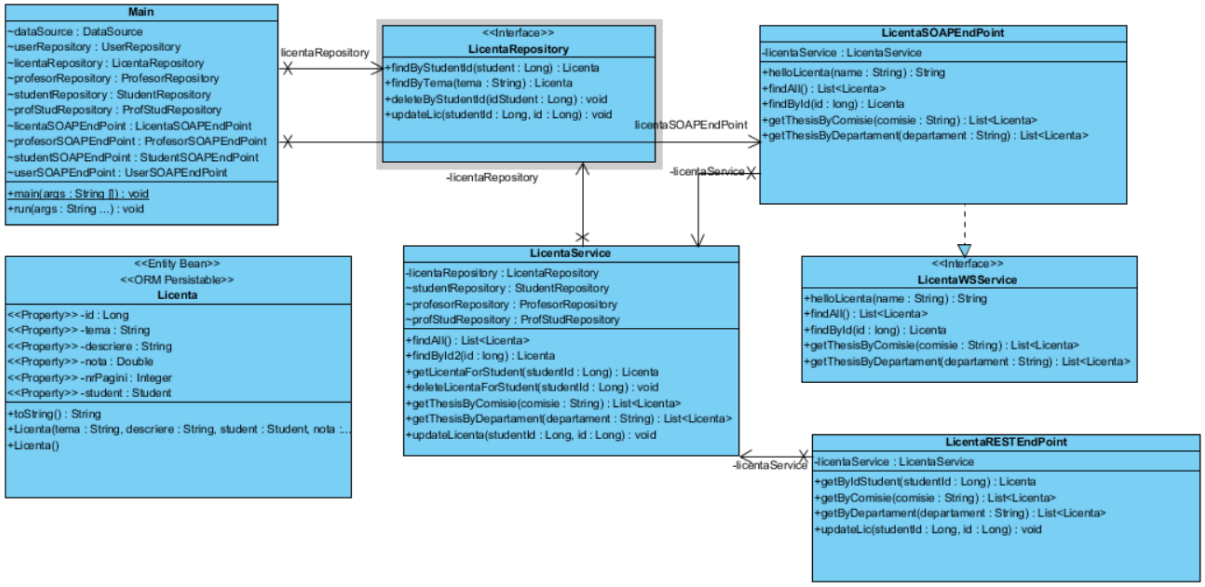


Figură 5-3 Diagrame claselor pentru operațiile pe entitatea Student



Figură 5-4 Diagrama claselor pentru operațiile pe entitatea *Profesor*

Următoarea figură conține prezentarea claselor necesare în implementarea operațiilor pe entitatea *Licență*.



Figură 5-5 Diagrama claselor pentru operațiile pe entitatea *Licență*

5.3. Interacțiunea dintre componente

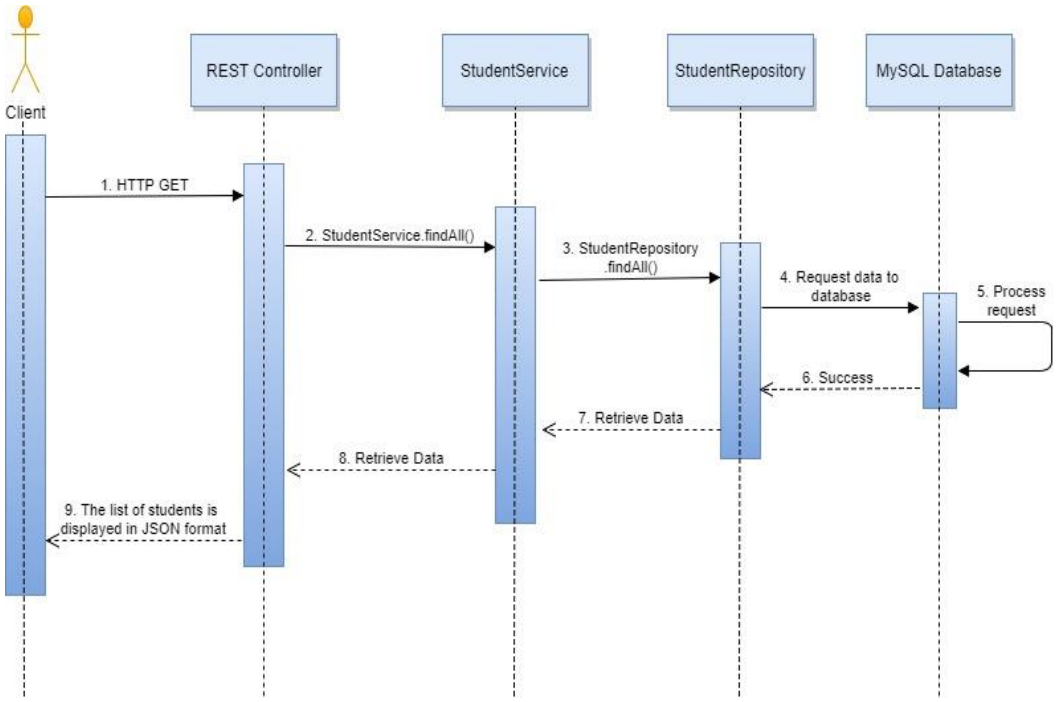
Acest subcapitol prezintă detalierea fluxului de comunicare între componentele prezentate, pentru a se satisface cazurile de utilizare precum: extragerea tuturor studenților înregistrați sau adăugarea unui student în listă. Operații se vor prezenta atât în contextul serviciilor RESTful, cât și al serviciilor SOAP.

În ceea ce privește serviciile RESTful, entru a se extrage toți studenții înregistrați, clientul trebuie să trimită o cerere de tip Get către adresa *localhost:8080/students/all-students*.Prin această cerere va apela funcția *findAll()* din clasa *StudentService*, care e definită și la nivel de *StudentRepository*, interfață ce extinde *CrudRepository* și prin care se extrag datele din baza de date MySQL. După ce datele au fost găsite în baza de date, acestea vor fi trimise într-o listă, urmând să ajungă la client în format JSON.

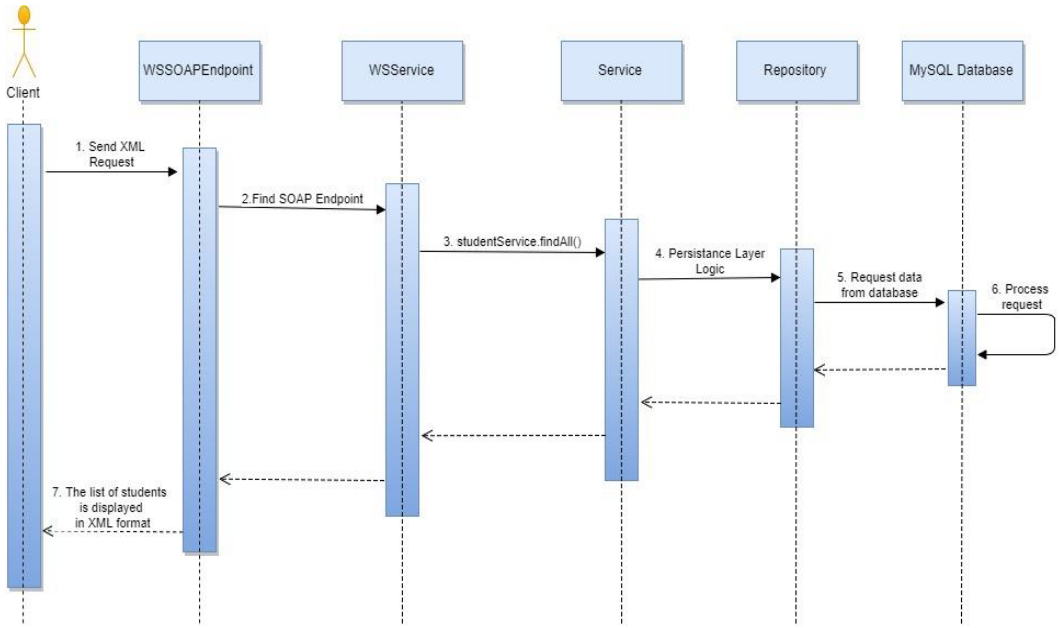
În cazul mesajelor de tip SOAP, dacă se dorește extragerea listei tuturor studenților înregistrați, clientul va trimite o cerere SOAP (în format XML) către adresa localhost:8081/wsdl/student.

După ce se va găsi metoda indicată și publicată prin intermediul wsdl-ului (în cazul de față metoda findAll() din studentService), aceasta va fi apelată la fel ca în cazul serviciilor RESTful. Spre deosebire de serviciile RESTful, în cazul SOAP mesajele ce vor fi transmise înapoi clientului vor avea formatul XML, iar parsarea mesajelor XML se realizează prin folosirea tehnologiei JAX-WS, descrisă în capitolul anterior.

Diagramele de secvență pentru cele două situații descrise în continuare:



Figură 5-6 Diagrama de secvență pentru găsirea tuturor studenților, în cazul RESTful



Figură 5-7 Diagrama de secvență pentru găsirea tuturor studenților, în cazul SOAP

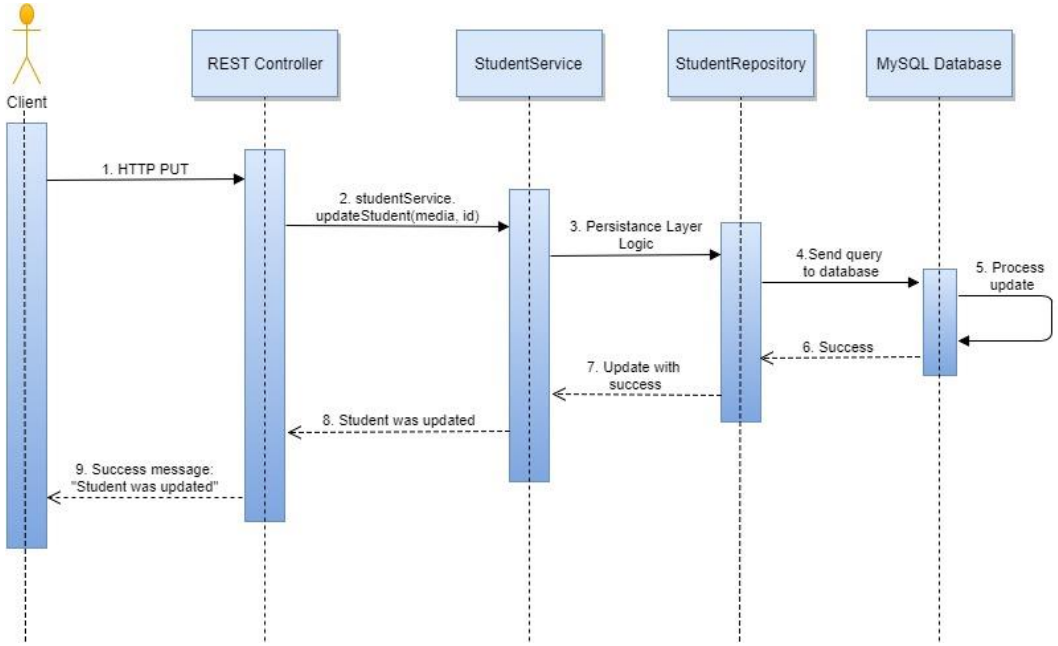
Un alt mod de implementare al operațiilor, folosit în aplicație, a fost folosirea interogărilor SQL în locul folosirii metodelor oferite de CrudRepository.

Astfel, în clasa ServiceRepository metoda implementată are adnotarile *@Modifying, @Transactional* și *@Query*, precum în exemplul atașat anterior:

```
@Modifying
    @Transactional
    @Query("UPDATE student SET media=?1 WHERE id=?2")
public void updateStudent(Double media, Long id);
```

Această metodă este apelată de servicile RESTful și SOAP exact ca și în cazul anterior, singura diferență constând în implementarea ei propriu-zisă, așa cum a fost descrisă anterior.

Pentru a exemplica modul de interacțiune al componentelor în această situație se prezintă figura 5.3.3. Ea este similară cu figura 5.3.1, cu diferența că cererea din baza de date se va face pe baza unei interogări SQL, iar metoda fiind de Update, cererea HTTP va fi de tip PUT. Exemplificarea este făcută în cazul REST, ea fiind similară cu diagrama 5.3.2 în contextul SOAP.

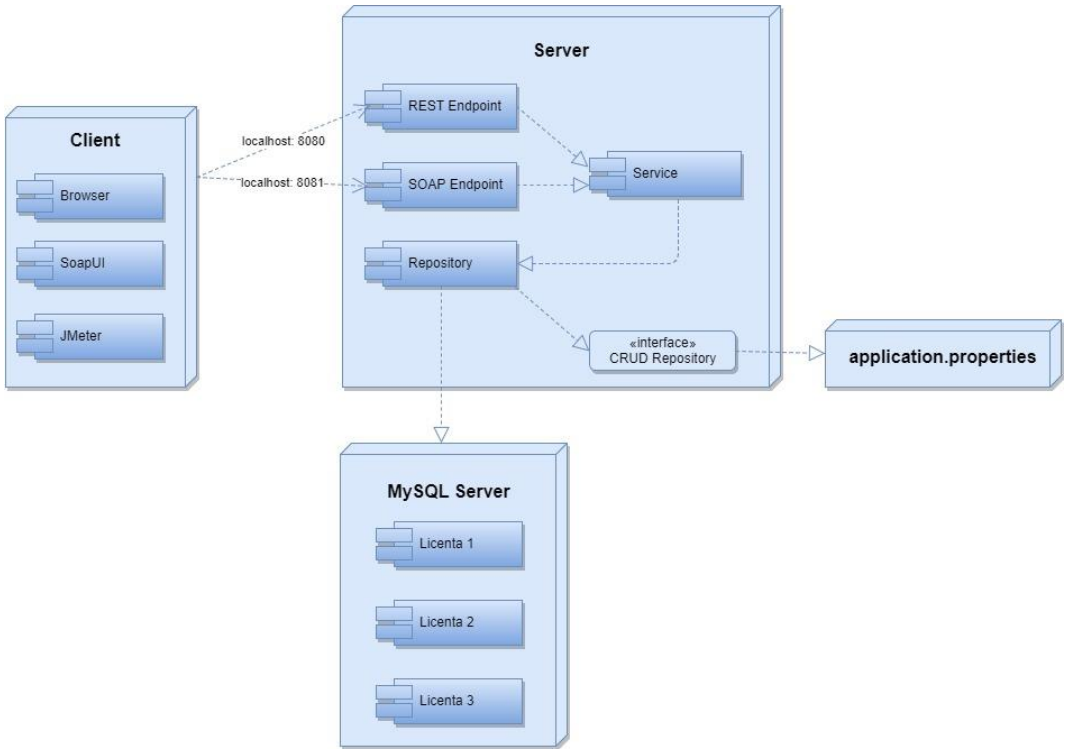


Figură 5-8 Diagrama de secvență pentru modificarea unui student, în cazul RESTful

5.4. Diagrama de deployment

Diagrama de deployment a aplicației prezintă componentele harware necesare utilizării sistemului, precum și modul în care acestea sunt interconectate.

Acestea vor fi prezentate în diagrama din figura 5.-9



Figură 5-9 Diagrama de deployment a aplicației

Componentele existente în aplicație, așa cum sunt prezentate și în diagrama anterioară, vor fi descrise în cele ce urmează. Acestea sunt:

Clientul – are rolul de a invoca serviciile web prin trimitere de cereri. Se remarcă 3 tipuri de clienți, și anume: ***Browser*** (spre exemplu browserul Chrome, folosit în trimiterea de cereri HTTP pentru serviciile RESTful), ***SoapUI*** (instrument dedicat testării și validării serviciilor RESTful și SOAP, care simulează trimitere de cereri HTTP sau SOAP și validează răspunsurile primite) și ***JMeter*** (instrument dedicat creării de teste de performanță pentru servicii RESTful sau SOAP).

Serverul – conține următoarele nivele: ***REST Endpoint, SOAP Endpoint, Service, Repository*** și ***CRUD Repository***.Atât componenta REST Endpoint, cât și SOAP Endpoint apelează metodele implementate în nivelul Service, care la rândul său implementează metodele definite în interfața Repository. Acesta, la rândul său, extinde interfața CrudRepository, implementând metodele CRUD definite la acest nivel.

Application.properties – reprezintă un fișier ce conține detalii de configurare privind accesul la baza de date. Spre exemplu, metoda de acces poate fi *create* sau *update*, în funcție de necesități. De asemenea, fișierul conține instanța bazei de date ce va fi folosită o dată cu rularea proiectului.

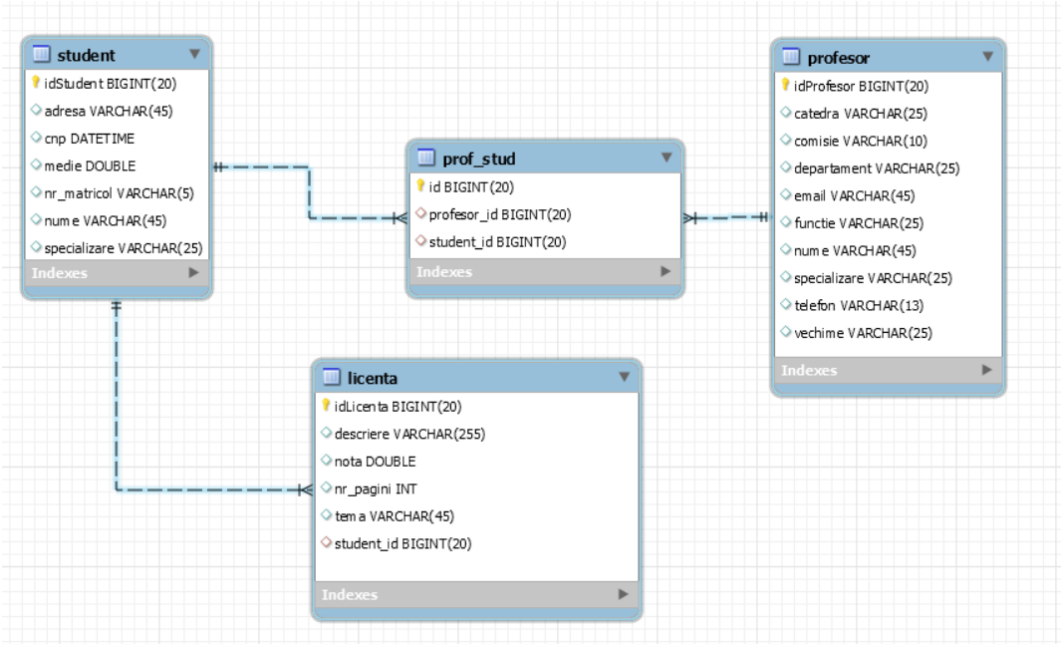
MySQL Server – reprezintă serverul bazei de date, care rulează pe portul 3306, prin intermediul instrumentului XAMPP Control Panel v3.2.2. După cum se observă și în figura 5.4, există 3 instanțe diferite ale bazei de date. Acest lucru se datoreaza faptului că, pentru analiza performanțelor serviciilor web, s-a avut în vedere varierea volumului datelor existente în baza de date.

5.5. Structura bazei de date

Baza de date utilizată este MySQL, fiind stocată cu ajutorului mediului MySQL Workbench. Baza de date este una simplă, conținând doar 4 tabele: Student, Profesor, Licenta și ProfStud (tabela de legătură între studenți și profesori). Accetul s-a pus pe volumul de date. În acest scop, s-a variat volumul bazei de date, astfel încât există 3 instanțe ale bazei de date:

- Licenta1db – conține 600 de studenți (fiecare cu o licență asignată) și 30 de profesori, ceea ce înseamnă că fiecărui profesor îi sunt asignați 20 de studenți.
- Licenta2db – conține 1800 de studenți (fiecare cu o licență asignată) și 30 de profesori, deci fiecărui profesor îi corespund 60 de studenți.
- Licența3db – conține 6000 de studenți (fiecare cu o licență asignată) și 30 de profesori, deci fiecărui profesor îi corespund 2000 de studenți.

Astfel, dacă se interoghează studenții de la comisia C1, aceasta având în componență 5 profesori, numărul studenților din lista rezultată va fi egal cu 5*20=100 în primul caz, respectiv 300 și 10000 de studenți. Aceste date vor fi utile în testele de performanță, când serverul va fi solicitat mult mai mult în a aduce 10000 de studenți în loc de 100, în situația inițială. Figura următoare prezintă diagrama bazei de date.



Figură 5-10 Diagrama bazei de date

Capitolul 6. Testare şi Validare

6.1. Testarea funcțională a serviciilor Web

Testarea funcțională a serviciilor Web are rolul de a garanta că operațiile au fost corect implementate, astfel încât un utilizator poate vizualiza datele cerute prin trimiterea unui request, fie el de tip HTTP sau de tip SOAP.

Pentru o testarea amănunțită a funcționalității serviciilor Web, s-au realizat cazuri de test, acestea specificând precondițiile ce trebuie respectate de utilizator, precum și pașii pe care acesta trebuie să-i execute spre a utiliza sistemul. În continuare se prezintă câteva cazuri de test, pentru flow-urile principale ale aplicației.

TC1: **Vizualizarea tuturor studenților înregistrați**

Precondiții: Clientul trebuie să dispună de conexiune la Internet

Descriere: Clientul poate vizualiza toți studenții înregistrați, validând datele acestora

Pasul 1: Utilizatorul deschide browser-ul Chrome

Pasul 2: Utilizatorul introduce URL-ul http://localhost:8080/students/all-students

Pasul 3 : Utilizatorul validează mesajul primit

Rezultatul așteptat: Un mesaj în format JSON este afișat în pagină, conținând lista tuturor studenților și datele referitoare la aceștia

TC2: **Vizualizarea tututor studenților la o anumită specializare**

Precondiții: Clientul trebuie să dispună de conexiune la Internet

Descriere: Utilizatorul poate vizualiza toți studenții aparținând unei anumite specializări

Pasul 1: Utilizatorul deschide browser-ul Chrome

Pasul 2: Utilizatorul introduce un URL folosind “query string”, ca în exemplul: http://localhost:8080/students/bySpecializare?specializare=TI

Pasul 3 : Utilizatorul validează mesajul primit

Rezultatul așteptat: Un mesaj în format JSON este afișat în pagină, conținând o listă de studenți și datele referitoare la aceștia

-Toți studenții din listă au specializare TI

TC3: Vizualizarea tuturor lucrărilor de licență din același departament

Descriere: Utilizatorul poate vizualiza toate licențele studenților care au ca îndrumător de licență profesori din același departament

Pasul 1: Utilizatorul deschide browser-ul Chrome

Pasul 2: Utilizatorul introduce un URL folosind “query string”, ca în exemplul: http://localhost:8080/thesis/byDepartament?departament=Calculatoare

Pasul 3 : Utilizatorul validează mesajul primit

Rezultatul așteptat: Un mesaj în format JSON este afișat în pagină, conținând o listă cu studenții așteptați și datele referitoare la aceștia. Toți studenții din listă au asignați profesori din departamentul de Calculatoare

TC4: Vizualizarea tututor lucrărilor de licență evaluate de aceeași comisie

Descriere: Utilizatorul poate vizualiza toate lucrările de licență a studenților asignați

profesorilor din aceeași comisie

Pasul 1: Utilizatorul deschide browser-ul Chrome

Pasul 2: Utilizatorul introduce un URL folosind "query string", ca în exemplul: http://localhost:8080/thesis/byComisie?comisie=C1

Pasul 3 : Utilizatorul validează mesajul primit

Rezultatul așteptat: Un mesaj în format JSON este afișat în pagină, conținând o listă cu licențele studenților și datele ale acestora.

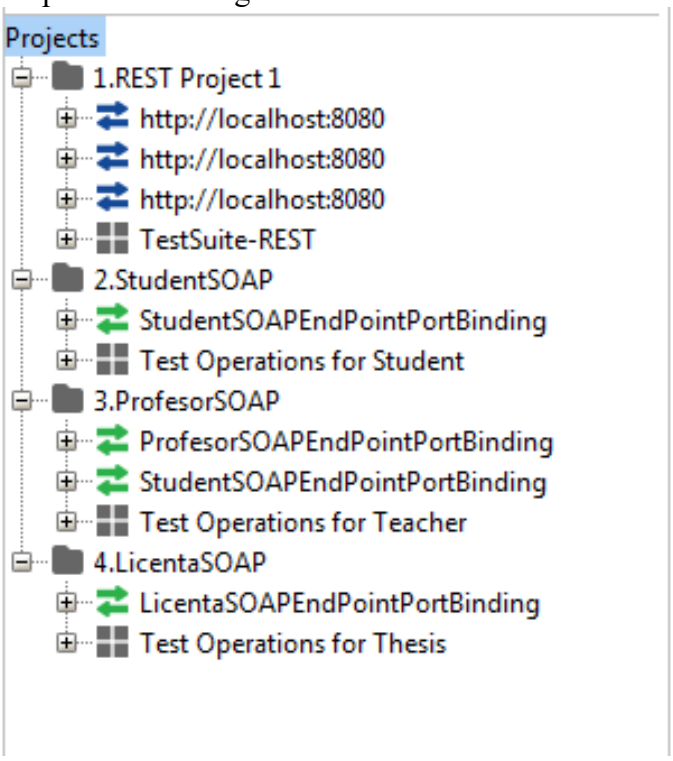
Testele manuale au fost realizate doar pentru serviciile REST, atât în cursul implementării lor, cât și la final, pentru a se verifica dacă funcționalitatea este cea dorită.

Pornind de la aceste scenarii de test, s-au realizat teste automate atât pentru serviciile RESTful, cât și pentru cele SOAP, cu ajutorului tool-ului SoapUI, oferit de SmartBear.

SoapUI oferă posibilitatea creării a două tipuri de proiecte (SOAP sau REST), fapt pentru care am realizat proiecte separate pentru a testa serviciile implementate.

Deoarece pentru SOAP am creat 3 WSDL-uri separate, e nevoie de 3 proiecte distincte, fiecare încărcând un WSDL propriu. Pentru a fi la fel de lizibil și pentru REST, au fost adăugate 3 resurse diferite, relevante pentru: Studenti, Profesori, respectiv Licențe.

Structura proiectelor este prezentată în figura alăturată:

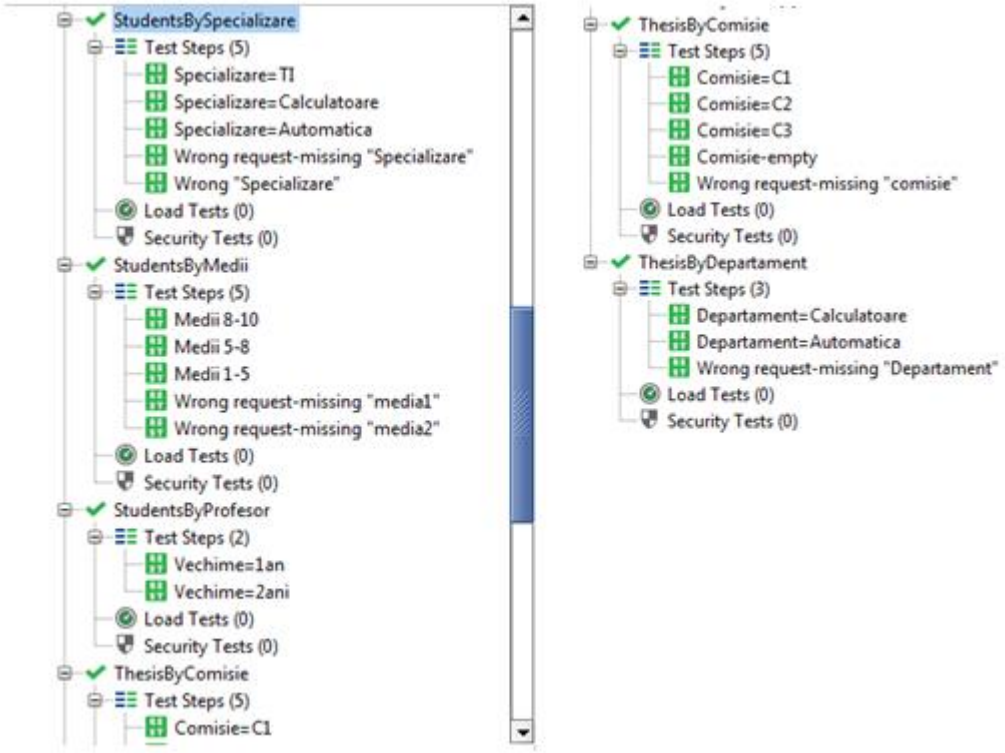


Figură 6-1 Proiectul REST în SoapUI

Pentru fiecare proiect a fost creată o suită de test, ce conține mai multe cazuri de test (Test Cases), formate din mai mulți pași. În cadrului proiectului de test pentru serviciile REST, a fost utilizat script-ul Groovy (limbaj de programare OOP) prin intermediul căruia se apelează cazurile de test automat, în funcție de scenariul de test dorit. Codul atașat constituie un script Groovy prin intermediul căruia se apelează testul pentru adăugarea unui student, iar apoi se apelează testul pentru vizualizarea tuturor studenților, validându-se că studentul adăugat anterior este prezent în lista curentă.

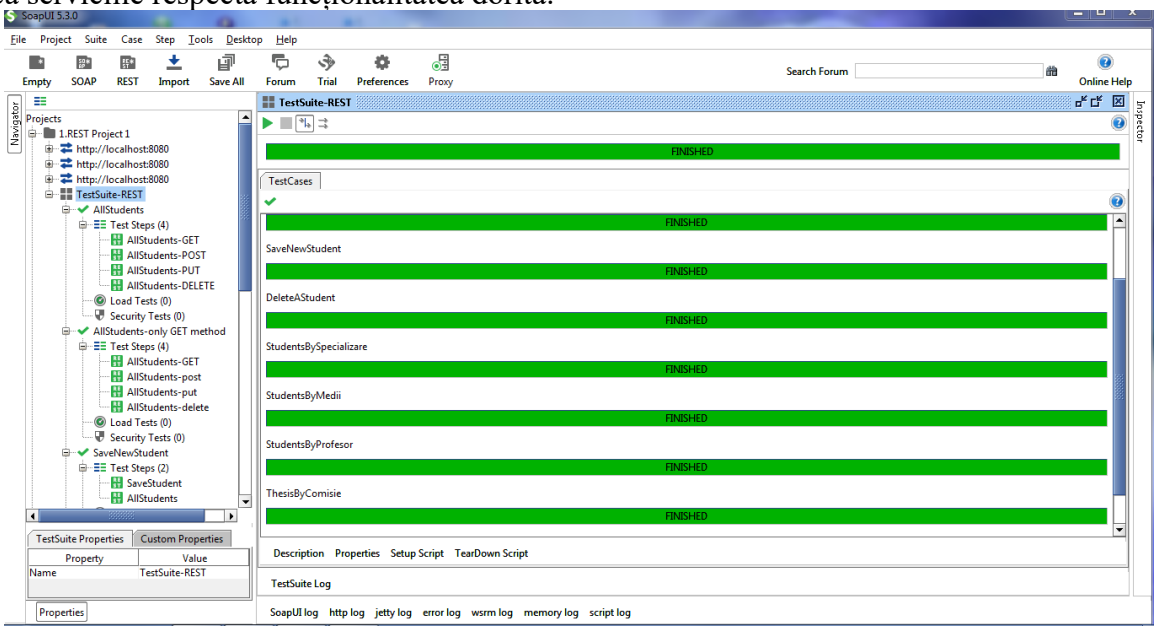
```
import groovy.json.JsonSlurper
//*********************************
//Add Student
def newStudent = '{ "student": {"nrMatricol":123, "nume":"Suciu Anca",
"cnp":2940617125471, "specializare":TI, "adresa":Str Plopilor, "media":8.2 } }'
def addStudentStep = testRunner.testCase.getTestStepByName("SaveStudent")
def addStudentRequest = addStudentStep.getTestRequest()

def x= addStudentRequest.setRequestContent(newStudent)
log.info(x)
testRunner.runTestStepByName("SaveStudent")
```

Testele sunt organizate în proiecte, acestea conținând suite de test alcătuite din mai multe cazuri de test, ce conțin fiecare unul sau mai mulți pași de test.



Figură 6-2 Detaliu al struturii unui proiect în SoapUI

În urma rulării întregii suite, rezultatele au fost cu succes, ceea ce reprezintă faptul că serviciile respectă funcționalitatea dorită.



Figură 6-3 Rezultatul testelor din SoapUI

Tabel 6-1 Analiza setului de operații

Numele operației	Numărul datelor aduse în urma unui request			Numărul tabelelor implicate
	LicentaDB1	LicentaDB2	LicentaDB3	
All Students	600	1800	6000	1
Add new student	1	1	1	1
Delete a student	1	1	1	1
Students BySpecializare	500; 100	1500; 300	5000; 1000	1
StudentsByMedii	1-600	3-1800	10-6000	1
Update student	1	1	1	1
All Teachers	30	30	30	1
Add new teacher	1	1	1	1
Delete a teacher	1	1	1	1
StudentsByTeacher	20	60	200	2
StudentsByComisie	120	360	1200	2
ThesisByStudent	1	1	1	3
ThesisByComisie	120	360	1200	3
ThesisBy Departament	400; 200	1200; 600	4000; 2000	3
Update thesis	1	1	1	1

6.2. Testarea de performanță

Testarea de peformanță are ca scop identificarea comportamentului unui sistem în condițiile reale, când este accesat de utilizatori concurenți. În situația de față, scopul testării performanțelor este de a compara serviciile RESTful și SOAP din punct de vedere al performanțelor acestora.

Pașii necesari în analiza performanței îi constituie studiul metricilor ce vor fi comparate, precum și operațiile asupra cărora se vor realiza testele de performanță.

Testarea nefiind exhaustivă, s-au ales pentru analiză operațiile care necesită mai multe resurse, acestea fiind considerate mai complexe. În tabelul următor se pot identifica aceste operații, printre lista tuturor operațiilor implementate.

Datele prezentate în tabel reflectă următoarele:

- Există 3 instanțe ale bazei de date, cu câte 600, 1800 și 6000 de studenți și 30 de profesori fiecare
- Fiecare student are asignată o singură licență
- Profesorii sunt împărțiți în mod egal în 5 comisii
- 20 dintre profesori aparțin departamentului „Calculatoare”, iar restul departamentului “Automatică”
- 500 dintre studenți au specializare “TI”, iar ceilalți “Calculatoare”

Operațiile pentru care se vor realiza teste de performanță sunt cele marcate în tabelul următor.

După acest studiu, se prezintă metricile de performanță ce vor rezulta în urma testelor construite cu JMeter și care trebuie analizate.

Metricile de performanță pe care le vom măsura sunt descrise în cele ce urmează:

- ***Average*** (reprezintă timpul mediu de răspuns, calculat în milisecunde)
- ***Median*** (valoarea de mijloc a timpului de răspuns dintr-o listă sortată de *samples*; Un median = 50% Line)
- ***90% Line*** (definește un timp de răspuns pentru care 90% dintre request-uri nu l-au depășit; Se calculează după formula: 90%Line=(90/100)*N+1/2, unde N=nr de *samples*)
- ***Min*** (timpul minim de răspuns)
- ***Max*** (timpul maxim de răspuns)
- ***Error %*** (Procentul request-urilor pentru care nu s-a obținut răspuns)
- ***Throughput*** sau ***Debit*** (Throughput=Numărul de request-uri/Timpul total de execuție)

Tabel 6-2 Rezultate obținute pentru DB1

Operație	Nr. Utilizatori-lor concu-renți	Request (ms)	Avg (ms)	Median (ms)	90% Line (ms)	Min (ms)	Max (ms)	Through-put (thread/ sec)
All Studens	100	REST	126	92	297	13	404	73.5
		SOAP	157	136	326	17	506	65.1
	500	REST	2388	2205	4632	59	6136	69.34
		SOAP	3245	3480	4568	110	5926	53.53
	1000	REST	4592	4855	6551	235	12670	44.2
		SOAP	6942	4831	12728	1685	14666	39.8
Thesis By Comisie	100	REST	2628	2768	3127	344	371	23.3
		SOAP	3843	3793	4305	318	496	13.7
	500	REST	33407	33193	48019	16128	53871	8.8
		SOAP	70242	70744	85923	49008	88151	5.6
	1000	REST	6988	5336	13428	2366	16297	17.9
		SOAP	26263	26850	31278	12055	38787	15.4
Thesis by Departa-ment	100	REST	6124	6362	6362	6990	2937	12.1
		SOAP	9757	9709	9757	10377	8839	6.9
	500	REST	22619	22564	36908	7492	39941	12.2
		SOAP	69581	69586	83547	52741	85541	5.8
	1000	REST	18322	14724	36365	7078	46113	6.6
		SOAP	77465	79048	94701	41662	12031	5.4

Tabel 6-3 Rezultate obținute pentru DB2

Operație	Nr. utilizatorilor concurenți	Request (ms)	Avg (ms)	Median (ms)	90% Line (ms)	Min (ms)	Max (ms)	Through-put (thread/ sec)
All Studens	100	REST	833	826	1381	36	2563	30.8
		SOAP	968	895	1624	87	1983	30.1
	500	REST	4487	3513	9024	88	14329	32.1
		SOAP	5524	5471	8179	731	12007	30.9
	1000	REST	48945	5241	195855	608	375254	1.25
		SOAP	130466	10214	280729	2202	374233	0.23
Thesis By Comisie	100	REST	6649	6718	7544	4558	8254	10.8
		SOAP	10528	10493	11099	9564	11824	7.0
	500	REST	25388	25189	41696	9309	46958	10.4
		SOAP	68249	68474	83993	47048	85424	5.8
	1000	REST	17650	15374	33335	6997	42320	6.4
		SOAP	76769	79121	96567	34001	129660	5.5
Thesis by Departa-ment	100	REST	22417	22455	23561	18020	24030	4.0
		SOAP	37019	37089	38143	34445	39020	2.3
	500	REST	63244	63021	104481	19724	113182	4.4
		SOAP	172916	180491	222350	30123	227554	2.2
	1000	REST	51037	44436	87200	18648	111009	2.3
		SOAP	223815	223663	273901	30121	370649	1.9

Tabel 6-4 Rezultate obținute pentru DB3

Operație	Nr. utilizatorilor concurenți	Re-quest (ms)	Avg (ms)	Median (ms)	90% Line (ms)	Min (ms)	Max (ms)	Through-put (thread/ sec)
All Students	100	REST	3250	3561	4425	943	6168	14.0
		SOAP	5399	6025	7338	2186	7613	9.3
	500	REST	11378	10229	24234	332	38652	12.6
		SOAP	16736	18190	24896	1000	34443	11.0
	1000	REST	41230	28307	58597	1464	302486	2.8
		SOAP	34054	26424	35408	5860	291647	1.8
Thesis By Comisie	100	REST	22411	22375	23635	20152	24483	4.0
		SOAP	33062	33030	34071	31390	34680	2.7
	500	REST	81165	84688	120678	25921	143976	3.4
		SOAP	128413	148716	208479	30093	240693	2.1
	1000	REST	53079	43743	104834	21554	130542	2.2
		SOAP	190406	176726	293229	30040	392546	2.0
Thesis by Departa-ment	100	REST	72904	73221	75022	68139	76219	1.3
		SOAP	85694	85684	86997	82642	89218	1.1
	500	REST	86924	78629	150507	31045	154430	3.2
		SOAP	94046	94727	99846	39041	823130	2.5
	1000	REST	12045	13107	174619	30038	197590	2.8
		SOAP	55362	38298	318740	30042	355452	2.3

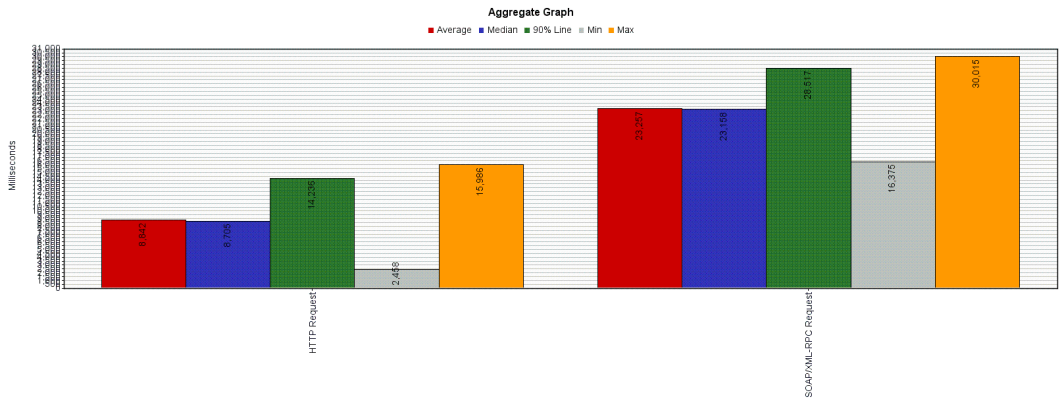
Pentru fiecare din operațiile menționate, s-au creat teste ce s-au rulat de câte 10 ori fiecare, alegându-se spre analiză acele rulări pentru care valorile metricilor erau medii.

După cum se observă și în tabelele anterioare, pentru fiecare operație s-au variat: volumul bazei de date și numărul de utilizatori concurenți.

În cele ce urmează se vor analiza rezultatele obținute în cazul operației *ThesisByComisie* , pentru fiecare din cele 3 instanțe ale bazei de date, denumite DB1, DB2 și DB3.

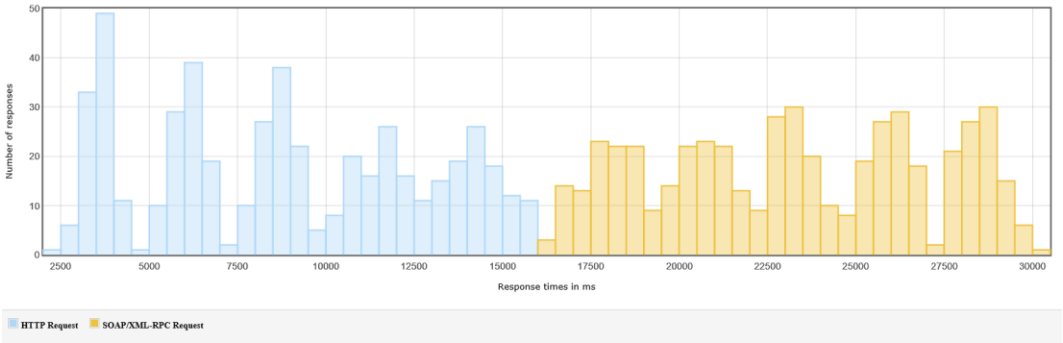
1. Operația *ThesisByComisie*, rulată pe DB1 cu 500 de utilizatori concurenți

În cazul operației *ThesisByComisie* rulată pe prima instanță a bazei de date, cu 500 de utilizatori concurenți, s-au obținut următoarele rezultate:



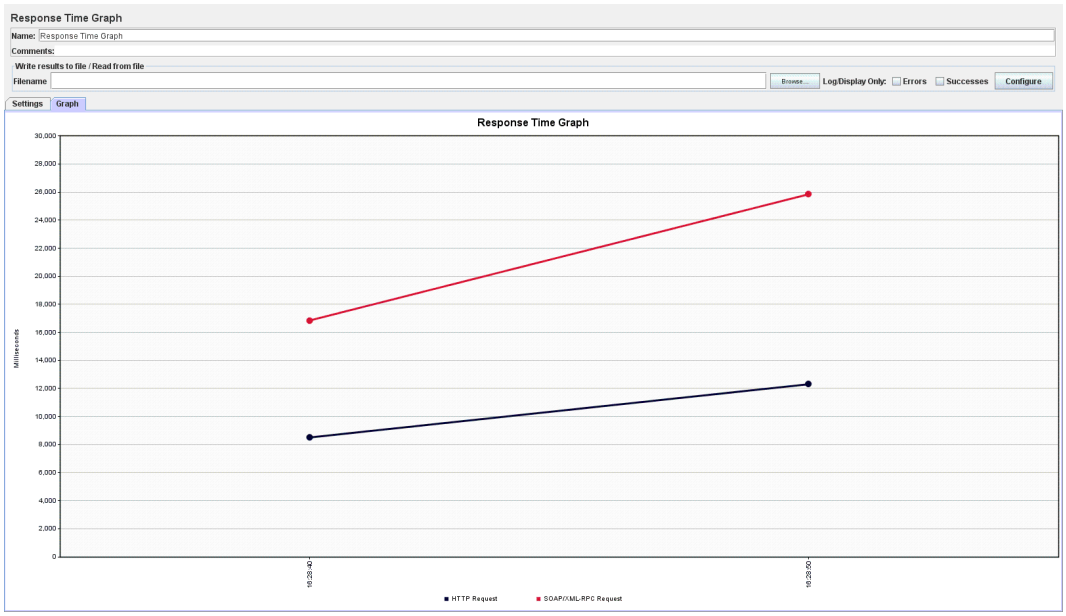
Figură 6-4 ThesisByComisie cu 500 utilizatori (DB1)

După cum se observă în graful prezentat anterior, rezultatele obținute au fost mult mai bune în cazul REST decât în cazul SOAP. Aceste valori exprimă timpul mediu de răspuns, medianul, procentul de 90% care nu va fi depășit, precum și timpul minim și maxim de răspuns pentru o cerere. Valorile pot fi identificate și în tabelul 6.2.
În continuare se prezintă distribuția firelor de execuție pe perioada rulării.



Figură 6-5 Distribuția firelor de execuție pentru ThesisByComisie (500 utilizatori, DB1)

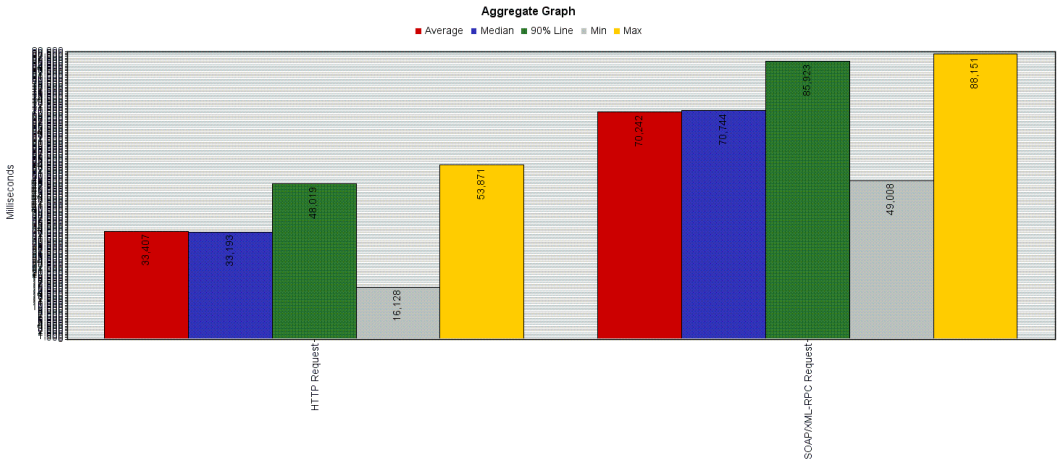
Timpul de răspuns obținut în urma rulării testului:



Figură 6-6 Timpul de răspuns pentru ThesisByComisie (500 utilizatori, DB1)

2. Operația *ThesisByComisie*, rulată pe DB2 cu 500 de utilizatori concurenți

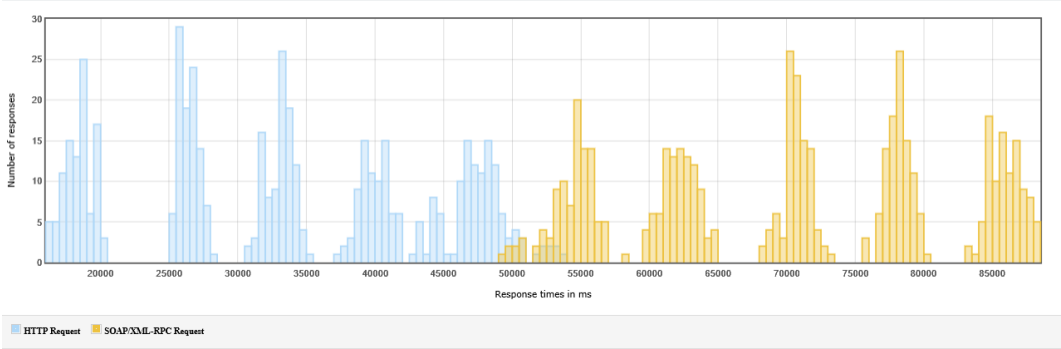
În cazul operației *ThesisByComisie* rulată pe DB2, cu 500 de utilizatori concurenți, s-au obținut următoarele rezultate:



Figură 6-7 ThesisByComisie cu 500 utilizatori (DB2)

După cum se observă în graful prezentat anterior, rezultatele obținute au fost mult mai bune în cazul REST decât în cazul SOAP. Aceste valori exprimă timpul mediu de răspuns, medianul, procentul de 90% care nu va fi depășit, precum și timpul minim și maxim de răspuns pentru o cerere. Valorile pot fi identificate și în tabelul 6.3

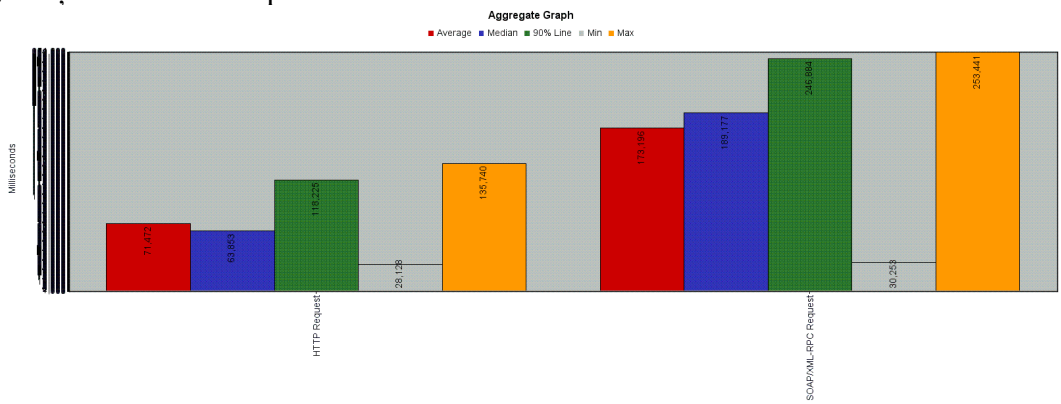
În continuare se prezintă distribuția firelor de execuție pe perioada rulării. După cum se poate observa, cererile de tip REST (*HTTP Request*) au avut timpul de răspuns bun mai bun ca cele de tip SOAP, astfel rezultând diferențele de performanță obținute.



Figură 6-8 Distribuția firelor de execuție pentru ThesisByComisie,500 utilizatori, DB2

3. Operația *ThesisByComisie*, rulată pe DB3 cu 500 de utilizatori concurenți

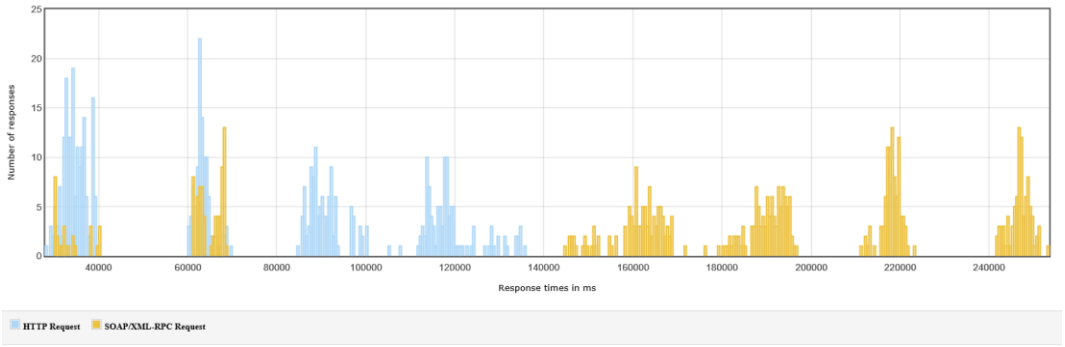
În cazul operației *ThesisByComisie* rulată pe DB2, cu 500 de utilizatori concurenți, s-au obținut următoarele rezultate. Graficul de mai jos este generat automat în JMeter, conținând metricile prezentate în tabelul 6.4.



Figură 6-9 ThesisByComisie cu 500 utilizatori (DB3)

După cum se observă în graful prezentat anterior, rezultatele obținute au fost mult mai bune în cazul REST decât în cazul SOAP. Spre exemplu, timpul mediu de răspuns (reprezentat cu roșu) este de 71,471 ms pentru REST și 173,196 ms pentru SOAP. Din figură se observă că toate valorile sunt mai mici pentru REST, indicând o performanță mai bună a acestuia.

În continuare se prezintă distribuția firelor de execuție pe perioada rulării. După cum se poate observa, cererile de tip REST (*HTTP Request*) au avut timpul de răspuns bun mai bun ca cele de tip SOAP, astfel rezultând diferențele de performanță obținute.

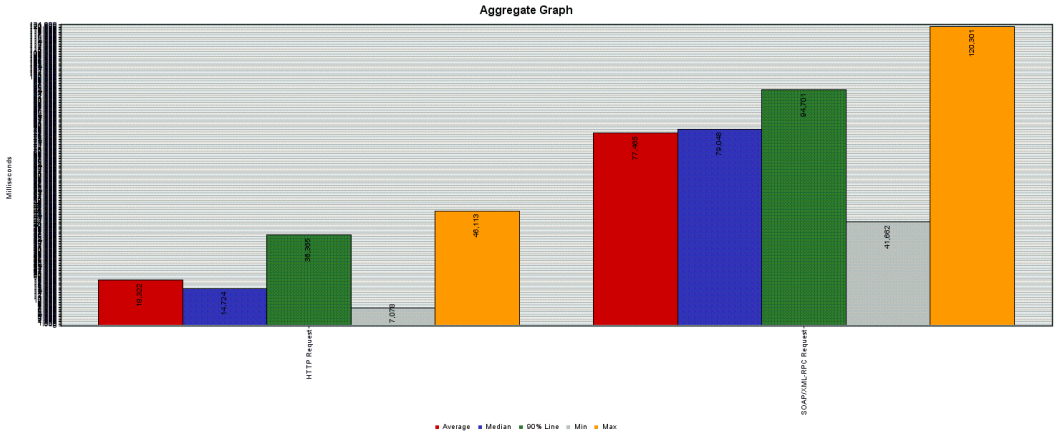


Figură 6-10 Distribuția firelor de execuție pentru ThesisByComisie,500 utilizatori, DB3

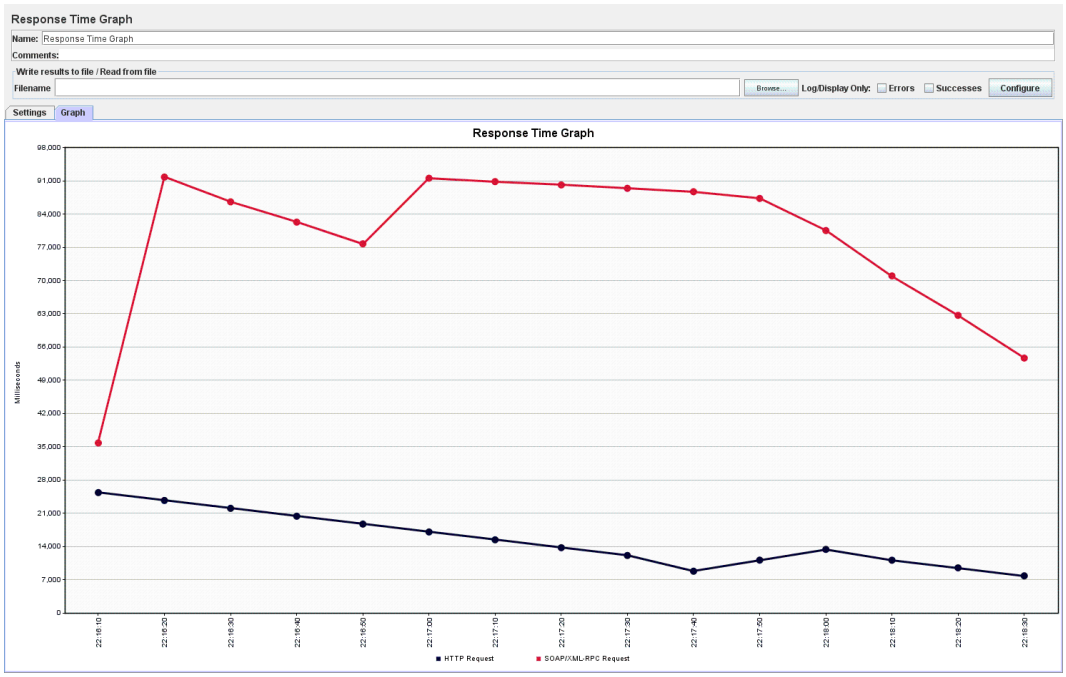
4.Operatia Thesis By Departament, rulată ce DB1 cu 1000 de utilizatori concurenți

Această operație a fost aleasă spre studiu fiind cea mai complexă (datorită numărului mare de date și a implicării tuturor tabelelor existente în baza de date). În continuare se vor prezenta rezultatele similar cu operațiile descrise anterior.

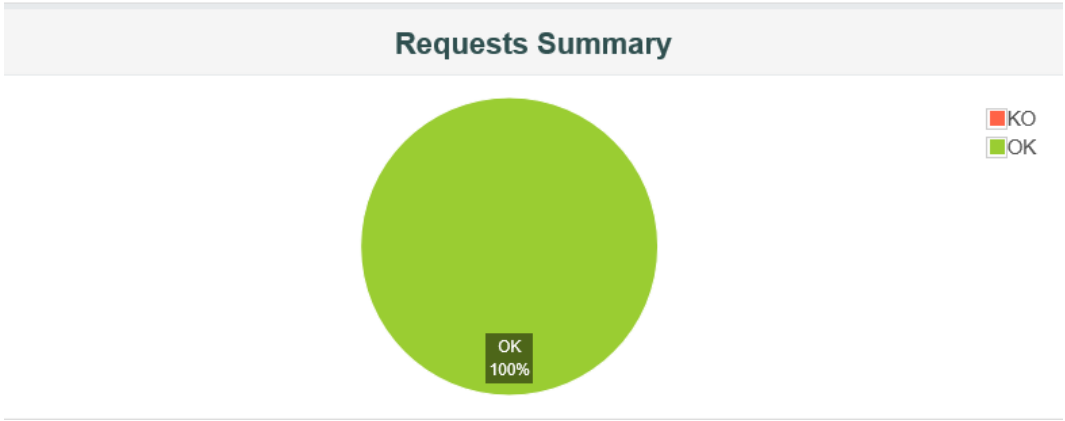
Rezultatele obținute în urma rulării:



Figură 6-11 Graful cumulativ cu parametrii obținuți

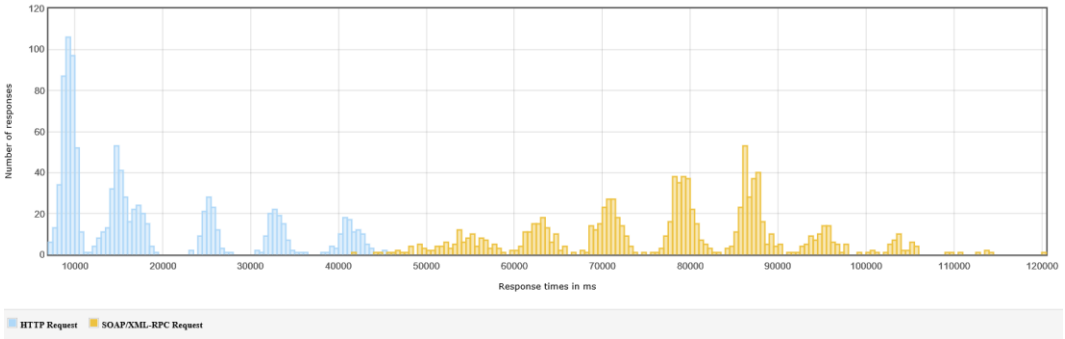


Figură 6-12 Graficul timpului de răspuns



Figură 6-13 Procentul cererilor cu success

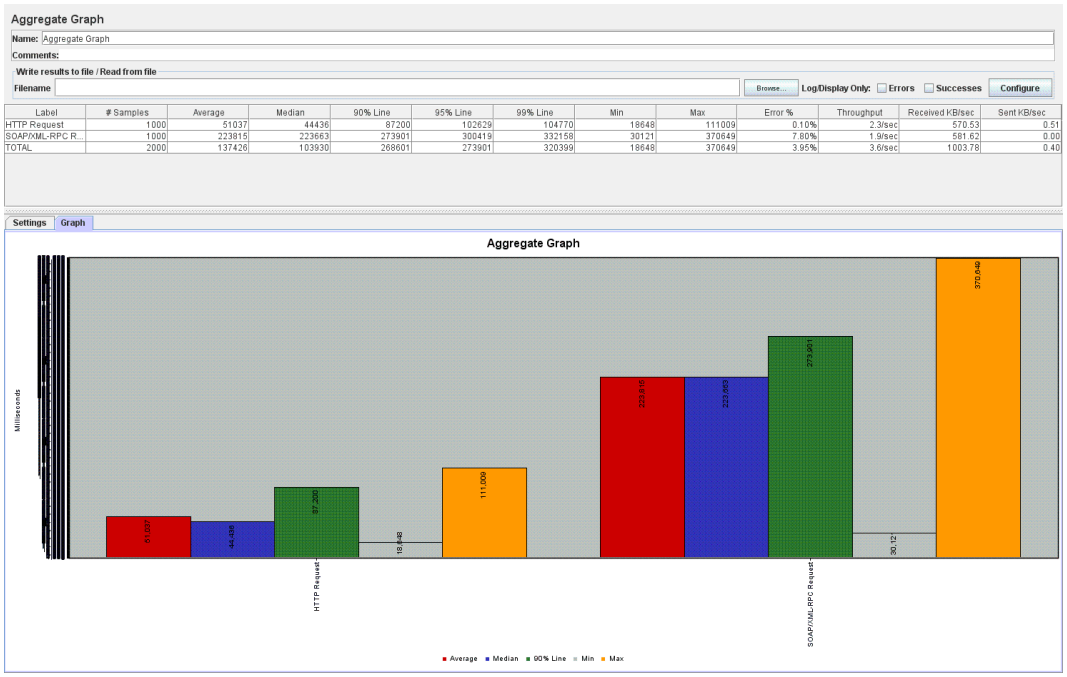
Imaginea antrioară prezintă faptul că toate cererile au fost executate cu success. În cazul în care serverul ar fi supra-solicitat și s-ar opri, cererile ar eșua.



Figură 6-14 Distribuția firelor de execuție

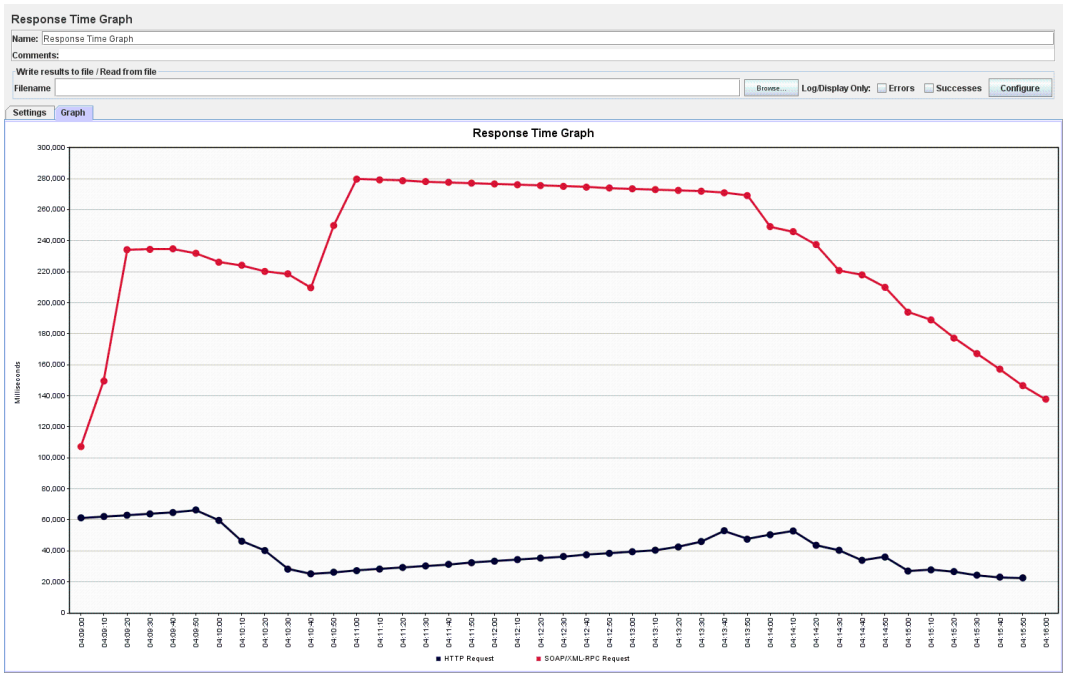
Distribuția firelor de execuție ilustrează faptul că cererile REST, reprezentate cu albastru au răspuns mult mai bine decît cererile REST.

5.Operatia ThesisByDepartament, rulată ce DB2 cu 1000 de utilizatori concurenți



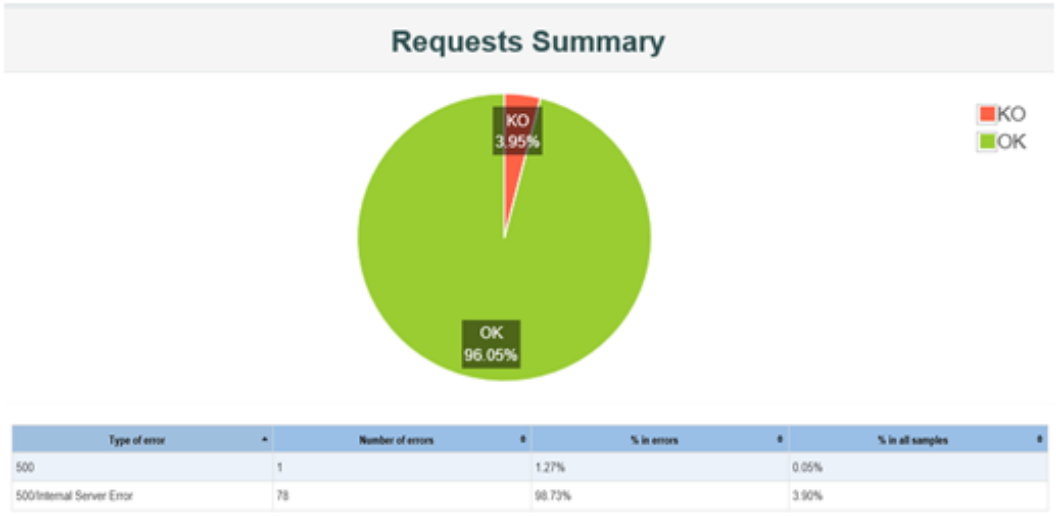
Figură 6-15 Grafic cumulativ al parametrilor

În graficul anterior sunt trecute valorile metricilor, obținute în urma rulării operației ThesisByComisie pentru DB2, cu 1000 de utilizatori. Un lucru important de observat este faptul că pentru o parte din cererile SOAP s-a obținut eroare, timpul de răspuns al acestora fiind mult peste cel așteptat. Timpul de răspuns este prezentat în figura următoare, ilustrând performanța mai bună a serviciilor REST în detrimentul celor SOAP.



Figură 6-16 Timpul de răspuns pentru ThesisByDepartament 1000 de utilizatori (DB2)

Erorile apărute pentru SOAP pot fi regăsite în figura de mai jos:



Figură 6-17 Procentul răspunsurilor cu succes/eșec

Capitolul 7. Manual de Instalare si Utilizare

Acest capitol este destinat prezentării pașilor necesari în instalarea sistemului, pentru a fi utilizat și analizat de un utilizator ulterior. Pașii necesari în instalarea sistemului sunt prezetați în cele ce urmează.

7.1. Instalarea utilitarului Eclipse Jee Neon

Acest instrument a fost folosit ca mediu de dezvoltare a aplicației, fiind necesar într-o utilizare ulterioară astfel încât orice stație să poată juca rolul de server care oferă serviciile web implementate. Java kit development împreuna cu Eclipse IDE for Java EE Developers pot fi descărcate și instalate folosind sursa: [16]

7.2. Instalarea utilitarului XAMPP Control Panel v3.2.2

XAMPP reprezintă instrumentul ce oferă serverul bazei de date MySQL și se găsește la sursa [25], de unde se recomandă a fi descărcat și instalat. După instalarea cu succes a programului, imediat la pornirea acestuia apar serviciile pe care acesta le oferă. Serverul MySQL este cel de care noi avem nevoie, iar pentru a se porni trebuie apăsat butonul de “Start” din dreptul său.

7.3. Instalarea MySQL Workbench

Pentru încărcarea bazei de date, se doreşte instalarea programului de gestiune a bazelor de date MySQL Workbench, disponibil pentru descărcare la adresa [26]. Instalarea acestuia presupune un proces iterativ uşor de realizat având intrucţiuni precise pe site-ul atașat anterior. La finalul instalării, se deschide utilitarul MySQL WorkBench, în care vom crea o conexiune către MySQL Server, care trebuie să aibă aceleaşi credenţiale ca şi în fişierul *application.properties* din arhiva proiectului. Dacă se creează o conexiune cu alt nume și altă parolă, aceste crendenţiale trebuie modificate pentru a coincide cu cele ale conexiunii.

În fișierul application.properties se specifică numele bazei de date, precum și modul de acces (*create* sau *update*). Se recomandă rularea proiectului cu modul *create* doar atunci când sunt apelate și metodele de citire din fișier, respectiv de scriere în baza de date.

7.4. Importarea și rularea proiectului

Pentru a importa proiectul în Eclipse, se deschide utilitarul şi se accesează *File>Import…-> Existing Maven Projects*, opţiune prin care se poate introduce calea către proiectul care se doreşte a fi importat, după care se selectează opţiunea “*Next*”. Astfel, proiectul va fi încărcat în workspace-ul curent al IDE-ului Eclipse Neon şi va fi pregătit de rulare, după ce se face build-ul acestuia, iar mai apoi se accesează pentru rulare.

7.5. Instalarea SoapUI

SoapUI reprezintă instrumentul folosit pentru testarea și validarea serviciilor web RESTful și SOAP, acesta fiind valabil la sursa [10]. Utilitarul poate fi descărcat din secțiunea "Download", existând atât versiune OpenSource, cât și versiunea SoapUI Pro. Versiunea OpenSource este cea folosită în proiect, fiind suficientă pentru realizarea testelor dorite de noi.

7.6. Importarea proiectului de test în SoapUI

Pentru crearea unui nou proiect în SoapUI, există două opțiuni diferite: *File>New SOAP Project* sau *File>New REST Project*, în funcție de dorința utilizatorului. Dacă se creează un proiect REST, cazurile de test vor fi create prin introducerea resursei (URL-ul) la care se va trimite cererea, precum și prin metoda HTTP dorită. În cazul unui proiect SOAP, acesta va fi creat prin introducerea WSDL-ului, în urma căruia se vor genera cererile în format XML.

Pentru a importa proiectele de test deja existente, pașii pentru proiectele REST și SOAP sunt aceiași. Se deschide *File>Import Project* și se alege proiectul dorit (ca de exemplu Test-Project.xml)

7.7. Instalarea JMeter și importarea proiectului de test

Aplicația Apache JMeter este, de asemenea, OpenSource și poate fi descărcată direct de pe site-ul producătorului, acesta fiind găsit la resursa [11]. Pașii necesari pentru instalare sunt ușor de realizat, fiind descriși la resursa prezentată. Pentru proiectul de față s-a folosit versiunea 3.1, dar poate fi utilă orice versiune mai recentă decât v3.0. JMeter reprezintă o aplicație dezvoltată 100% în limbajul de programare Java, iar după decărcarea acesteia, un executabil denumit ApacheJMeter se va găsi în folderul *apache-jmeter-3.1>bin*, care după ce va fi rulat va deschide aplicația desktop prezentată. Pentru a importa un proiect de test în JMeter, pașii sunt simpli, fiind similari cu cei din Eclipse: File>Open>Test_Plan_Licenta.jmx. Odată ce proiectul a fost importat, se va deschide planul de test ce va cuprinde mai multe "Thread Group-uri", fiecare reprezentând o operație testată.

7.8. Configurarea JMeter pentru generarea unui raport în HTML

Pentru a genera un raport HTML, sunt necesari următorii pași:

-Editarea fișierului *user.properties* prin adăugarea elementelor ce se doresc a fi analizate (spre exemplu, pentru timp se adaugă *jmeter.save.saveservice.time=true*)

-Rularea unui test și salvarea rezultatelor într-un CSV file, prin folosirea unui Listener de tip Simple Data Writer

-Generarea fișierului HTML pornind de la CSV-ul creat, prin rularea comenzii *jmeter -g path_of_input_file.csv -o path_of_html_report* în linia de comandă.

Capitolul 8. Concluzii

În acest capitol se vor prezenta succint obiectivele inițiale, realizările care rezultă în urma dezvoltării acestei lucrări, precum și posibile dezvoltări ulterioare.

8.1. Realizările obiectivelor propuse

- Această lucrare a avut ca principal scop compararea serviciilor Web de tip RESTful, respectiv SOAP. Astfel, primul pas a constat în crearea unui context de aplicație, realizat prin implementarea unui set de operații comune pentru REST și SOAP.
- În acest scop, a fost implementat un sistem de management al situației școlare a studenților din an terminal, prin intermediul căruia se pot extrage: lista de studenți, lista licențelor înscrise, studenții ce vor fi examinați de către o anumită comisie, profesorii dintr-un anumit departament, licențele ce vor fi prezentate unei anumite comisii și așa mai departe.
- Operațiile descrise anterior au fost implementate atât în REST, cât și în SOAP, dorindu-se comparația acestora din punct de vedere al performanței. După ce au fost validate din punct de vedere funcțional, s-au ales cele mai complexe operații pentru a se realiza teste de performanță asupra acestora.
- Astfel, s-au testat operațiile variind atât volumul bazei de date,cât și numărul de utilizatori concurenți . Cererile de tip REST și SOAP au fost trimise prin simularea mai multor utilizatori concurenți, unii trimițând cereri HTTP, iar ceilalți cereri de tip SOAP.

8.2. Analiza conceptuală a serviciilor web

Avantajele REST în detrimentul SOAP:

- În plus față de utilizarea HTTP pentru simplitate, REST oferă o serie de alte avantaje față de SOAP:
- REST permite o mai mare varietate de formate de date, în timp ce SOAP permite doar XML.
- Cuplat cu JSON (care de obicei funcționează mai bine cu datele și oferă o analiză mai rapidă), REST este, în general, considerat mai ușor de utilizat.
- Datorită JSON, REST oferă un suport mai bun pentru clienții browserului.
- Protocolul este utilizat cel mai adesea pentru servicii importante cum ar fi Yahoo, Ebay, Amazon și chiar Google.
- REST este în general mai rapid și utilizează mai puțină lățime de bandă. Este, de asemenea, mai ușor de integrat cu site-urile existente fără a fi nevoie să se refacă infrastructura site-ului.

Avantajele SOAP în detrimentul REST:

- În unele cazuri, proiectarea serviciilor SOAP poate fi mai puțin complexă în comparație cu REST. Pentru serviciile web care suportă operații complexe, care necesită menținerea conținutului și a contextului, proiectarea unui serviciu SOAP necesită mai puțină codificare în stratul aplicației pentru tranzacții, securitate, încredere și alte elemente.
- SOAP este foarte extensibil prin alte protocoale și tehnologii.
- În plus față de WS-Security, SOAP acceptă WS-Addressing, WS-Coordination, WS-ReliableMessaging și o serie de alte standarde de servicii web.
- SOAP este potrivit pentru tranzacții compatibile cu ACID, precum și în cazurile de utilizare ce necesită fiabilitate tranzacțională mai mare decât cea care se poate obține prin HTTP (ce limitează REST în această capacitate).

Deoarece se pot obține, de cele mai multe ori, funcționalități similare folosind REST sau SOAP, decizia alegerii uneia dintre aceste tehnologii este subiectivă și depinde de contextul utilizării.

8.3. Analiza rezultatelor obținute

În urma mai multor rulări consecutive, rezultatele au arătat faptul că REST are performanțe mai bune, timpul de răspuns la o cerere fiind de cele mai multe ori, mult mai mic decât în cazul SOAP.

- Cu cât crește numărul de utilizatori concurenți, timpul de răspuns este mai mare, iar debitul se micșorează.
- Cu cât crește volumul bazei de date, se observă o scădere a performanței atât în cazul SOAP, cât și REST, cel din urmă fiind încontinuu mai rapid decât precedentul.
- În cazul în care se supra-solicită serverul, cererile SOAP primesc erori mai repede decât cererile REST.
- REST se remarcă printr-un debit mai mare decât cel obținut de SOAP, ceea ce reprezintă un alt indicator al performanței înalte, întrucât utilizează mai puțină lățime de bandă.
- Unul din motivele pentru care REST este mai performant e dat de faptul că REST folosește mesaje în format JSON, acestea fiind mult mai ușor de procesat decât mesajele în format XML, cu care lucrează SOAP. REST utilizează mai puțină lățime de bandă, motiv pentru care este de multe ori de dorit.

8.4. Dezvoltări ulterioare

- Ca orice sistem informatic, aplicația prezentată poate beneficia de unele dezvoltări ulterioare, atât în ceea ce privește cerințele funcționale, cât și cerințele non-funționale.
- De asemenea, diverse îmbunătățiri ar putea fi aduce la nivelul benchmark-ului creat în scopul testării performanței, analizei și comparării metricilor de performanță.
- În cele ce urmează voi prezenta o listă cu posibile dezvoltări ulterioare:
    - Dezvoltarea unor operații mai complexe spre a fi comparate
    - Adăugarea mai multor aserții non-funționale în testele de performanță, pentru a urmări o eventuală îmbunătățire sau o eventuală descreștere a performanței în timp
    - Rularea serviciilor de pe o mașină virtuală, astfel încât rularea testelor să nu fie influențată de alte procese
    - Testarea serviciilor pe un alt sistem de operare, pentru a studia comportamentul serverului în alte condiții
    - Crearea unei copii a bazei de date, astfel încât fiecare dintre servicii să acceseze propria bază de date, ceea ce ar evita apariția unui bottleneck la nivelul bazei de date
    - Integrarea testelor existente cu Jenkins, pentru a putea fi monitorizate constant
    - Simularea a câte unui JVM separat pentru REST și SOAP, astfel încât s-ar evidenția comportamentul fiecăruia în parte, nefiind influențat de alte procese

Bibliografie

[1] E. Cerami, Web Services Essentials, Distributed Applications with XML-RPC,SOAP,UDDI&WSDL, O'Reilly, 2002.

[2] A. H. N. Z. Yutu Liu, „QoS Computation and Policingi in Dynamic Web Services Selection," p. 8, 2004.

[3] L. Cherkasova, „Measuring and Characterizing End-to-End Internet Service Performance," New York, 2003.

[4] P. K. Potti, „On the Design of Web Services: SOAP vs. REST," North Florida, 2011.

[5] A. Richardson, Automating&Testing a REST API, Leanpub, 2017.

[6] SmartBear, „SoapUI by SmartBear," disponibil la: https://www.soapui.org.

[7] My SQL, „MySQL Workbench Manual," Disponibil la: https://dev.mysql.com/doc/workbench/en/wb-installing-windows.html.

[8] Apache Friends, „XAMPP Apache + MariaDB + PHP + Perl," Disponibil la: https://www.apachefriends.org/index.html.

[9] Eclipse, „Eclipse packages," disponibil la: http://www.eclipse.org/downloads/eclipse-packages/.

[10] SoapUi by SMARTBEAR, „Build Better | Test Smarter The Most Advanced REST&SOAP Testing Tool in the World," disponibil la: https://www.soapui.org/.

[11] The Apache Software Foundation, Apache JMeter™, disponibil la: http://jmeter.apache.org/.

[12] B. S. Alboaie Lenuța, Servicii Web: concepte de bază și implementări, Iași: Editura POLIROM, 2006.

[13] J. V. D. S. Michael zur Muehlen, Developing Web Services Choreography Standards-The case of REST vs SOAP, Elsevier B.V., 2004.

[14] Pagina oficiala Oracle, https://www.oracle.com/index.html.

[15] Documentatia oficiala Eclipse, disponibil la: http://help.eclipse.org/oxygen/index.jsp.

[16] Descarcare pachete Eclipse, disponibil la: http://www.eclipse.org/downloads/eclipse-packages/.

[17] Documentatia oficiala Apache Tomcat, disponibil la: http://tomcat.apache.org/.

[18] Documentatia oficiala Apache Maven Project, disponibil la: http://maven.apache.org/.

[19] Documentatie si tutoriale JPA, disponibil la: https://www.tutorialspoint.com/jpa/.

[20] Documentatie de referinta - Spring Data JPA, disponibil la: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.

[21] Documentatie JAX-WS si JAX-RS, disponibil la: http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html.

[22] Documentatie JAXB, disponibil la: http://www.oracle.com/technetwork/articles/javase/index-140168.html.

[23] Documentatie oficiala Apache Groovy, http://groovy-lang.org/.

[24] Documentatie oficiala Apache JMeter, http://jmeter.apache.org/.

[25] Documentatie oficiala Apache Friends - XAMPP Apache + MariaDB + PHP + Perl, disponibil la:

https://www.apachefriends.org/index.html.

[26] Documentatie oficiala MySQL Workbench, disponibil la: https://dev.mysql.com/doc/workbench/en/workbench-faq.html.

[27] Salomie, T. Cioară, I. Anghel, T. Salomie, Distributed Computing and Systems. A Practical Approach, Editura Albastră, 2008

Anexa 1 - Glosar de termeni

API	Application Programming Interface
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SOA	Service Oriented Architecture
WSDL	Web Service Description Language
XML	EXtensible Markup Language
UDDI	Universal Description, Discovery and Integration
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
CRUD	Create, Read,Update,Delete
J2EE	Java 2 Enterprise Edition
JPA	Java Persistence API
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
SQL	Structured Query Language
XAMPP	Cross-Platform, Apache, MariaDB, PH and Perl
JAXB	Java Architecture for XML Binding
JAX-WS	Java API for XML Web Services

Anexa 2 – Lista figurilor

Anexa 3 – Lista tabelelor