



**Universitatea Transilvania din Braşov**

**Facultatea de Matematică şi Informatică**

**Specializarea Informatică aplicată**

**Aplicaţie web folosită în dezvoltarea  
unui magazin de vânzări online**

**Autor**

**Iuga Delia Elena**

**Coordonator ştiinţific**

**Conf.univ.dr.Silviu Răzvan Dumitrescu**

**Brasov  
2014**

# Aplicaţie web folosită în dezvoltarea unui magazin de vânzări online

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>5</b>
1.1	Evoluția cumpărăturilor . . . . .	5
1.1.1	Mai multe despre e-commerce . . . . .	6
1.2	Motivarea temei . . . . .	7
1.3	Soluții actuale . . . . .	8
<b>2</b>	<b>Aplicații web</b>	<b>10</b>
2.1	Despre aplicații web . . . . .	10
2.2	Arhitectura client-server . . . . .	11
2.3	Protocolul HTTP . . . . .	12
2.3.1	Tranzacție HTTP . . . . .	13
2.4	Cum funcționează aplicațiile web? . . . . .	14
<b>3</b>	<b>Unelte de dezvoltare</b>	<b>16</b>
3.1	Sistem de versionare.Tortoise SVN . . . . .	16
3.2	Maven . . . . .	18
<b>4</b>	<b>Persistarea datelor</b>	<b>20</b>
4.1	Persistarea datelor în baze de date . . . . .	20
4.1.1	MySQL . . . . .	21
4.2	JPA . . . . .	23
4.2.1	Entități . . . . .	24
4.2.2	Relaționarea entităților . . . . .	25
4.3	Hibernate . . . . .	26
<b>5</b>	<b>Web server vs Application server</b>	<b>27</b>
5.1	JBoss 7.1.1 AS . . . . .	28
<b>6</b>	<b>View</b>	<b>29</b>
6.1	JSF . . . . .	29
6.1.1	Beneficii JSF . . . . .	29

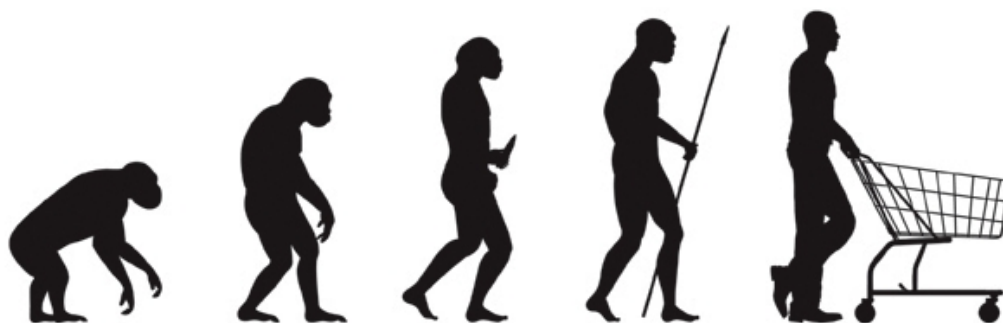
6.1.2	Arhitectura JSF . . . . .	30
6.1.3	Ciclul de viaţă al unei aplicaţii JSF . . . . .	30
6.2	Managed Beans . . . . .	31
6.2.1	Adnotări . . . . .	31
6.3	Primefaces . . . . .	32
6.3.1	XHTML . . . . .	33
6.3.2	AJAX . . . . .	33
6.3.3	JavaScript . . . . .	34
6.4	CSS . . . . .	35
<b>7</b>	<b>Controller</b>	<b>36</b>
7.1	EJB . . . . .	36
7.2	JavaMail . . . . .	38
7.3	IText . . . . .	39
7.4	JSON . . . . .	40
7.4.1	GSON . . . . .	40
7.5	JAXB . . . . .	41
7.6	Servicii REST . . . . .	42
7.6.1	Dezvoltarea serviciilor REST folosind JAX-RS . . . . .	43
7.7	Comet . . . . .	44
<b>8</b>	<b>Structura aplicaţiei.MVC</b>	<b>45</b>
<b>9</b>	<b>Aplicaţia rezultată</b>	<b>47</b>
9.1	Prezentarea bazei de date <b>Shop4j</b> . . . . .	47
9.2	Funcţionalităţi . . . . .	52
9.2.1	Schimbare cont . . . . .	53
9.2.2	Vizualizare clienţi . . . . .	55
9.2.3	Generare rapoarte . . . . .	56
9.2.4	Actualizare stoc produse . . . . .	58
9.2.5	Recuperare parola . . . . .	60
9.2.6	Livrare comenzi . . . . .	62
9.2.7	Comenzi returnate . . . . .	64
9.2.8	Mesajele clienţilor . . . . .	66
9.2.9	Chat . . . . .	68
<b>10</b>	<b>Concluzii</b>	<b>70</b>

# Capitol 1

## Introducere

### 1.1 Evoluția cumpărăturilor

Shopping-ul a existat, într-o formă sau alta, de când omul a apărut pe pământ. Am trecut de la a răsfoi cataloage de cumpărături la a face o comandă apăsând un singur click. Obiceiul nostru de a cumpăra se schimbă pe măsură ce tehnologia avansează.



O scurtă prezentare a locurilor unde oamenii își făceau cumpărăturile de-a lungul timpului:

1. **Bazare:** dintre toate, se evidențiază *Marele bazar din Istanbul* care rămâne până astăzi cel mai mare centru de cumpărături acoperit din lume (250000-400000 vizitatori pe zi: conform [9]).
2. **General stores (magazinele generale):** serveau ca punct central de întâlnire pentru orașenii și fermierii care erau în căutare de hrană, îmbrăcăminte și semințe pentru plante.

3. **Cataloage de cumpărături prin poştă:** Pe la sfârşitul anilor 1800, mai mult de jumătate din locuitorii Statelor Unite locuiau în zone rurale şi drumul până la magazinele situate în oraş era destul de istovitor. Vanzatorii au găsit o nouă modalitate de a-şi vinde produsele şi anume *catalogul de comandă prin poştă*.
4. **Mall-urile:** Îmbunătăţirea sistemelor de autostradă, extinderea aşezărilor urbane şi popularitatea automobilelor au condus la dispersia cumpărăturilor departe de centrele urbane, precum şi la creşterea numărului de mall-uri.
5. **Cum facem cumpărături în prezent:** Cu din ce în ce mai mult acces la internet, e-commerce a luat cu asalt industria de retail.

Să luăm ca şi exemplu **e-bay** care combină sisteme vechi de când lumea ca trocul şi tocmeala cu e-commerce-ul din zilele noastre triumfând ca primul site de licitaţii din lume.[9]

**2.457 de bunuri licitate în fiecare secundă pe e-bay**

=

**147.240 \$ pe minut**

### 1.1.1 Mai multe despre e-commerce

E-commerce, sau pe româneşte, *comerţul electronic* se referă la comercializarea de produse sau realizarea de servicii prin intermediul reţelelor de calculatoare(*computer networks* mai exact), cum ar fi *Internetul*. Comerţul electronic modern, foloseşte de obicei *World Wide Web*, cel puţin la un moment dat în ciclul de viaţă al tranzacţiei, deşi acesta poate cuprinde o gamă mai largă de tehnologii, cum ar fi e-mail, dispozitive mobile, social media, şi telefoane de asemenea.[10]

*Online shopping* sau e-shopping-ul este o formă de comerţ electronic, care permite consumatorilor să cumpere bunuri sau servicii de la un vânzător de pe Internet folosind un *browser web*. Cumpărătorii pot astfel să viziteze magazine on-line în confortul fotoliului preferat şi să cumpere stând în faţa calculatorului.

Avantajele cumpărăturilor on-line:[11]

- |  |                                  |
|--|----------------------------------|
| 1. <b>Convenienţa</b>                  | 2. <b>Preţuri bune</b>           |
| 3. <b>Alegeri infinite</b>             | 4. <b>Vizualizare comentarii</b> |
| 5. <b>Fără presiune la cumpărături</b> | 6. <b>Achiziţii discrete</b>     |

## 1.2 Motivarea temei

Proiectul de faţa îmbină 2 mari concepte şi anume *cumpărăturile* şi *aplicaţiile web*.

Cum am menţionat în secţiunea Evoluţia cumpărăturilor, cumpărăturile sunt în strânsa legătură cu tehnologia: *obiceiul de a cumpăra se schimbă pe măsură ce tehnologia avansează*. Şi nici nu ar putea fi altfel; dacă am lăsa ca marile descoperiri să treacă pe lângă noi fără să să profităm de beneficiile pe care le aduc şi fără să ne adaptăm obiceiurile la schimbările pe care acestea le presupun, atunci nu am cunoaşte niciodată evoluţia;

Data fiind importanţa necesităţii unei aplicaţii care să simuleze un magazin online, este evident că mulţi programatori au încercat, şi reuşit să implementeze un astfel de soft; deci, imediat apare întrebarea 'de ce am ales să fac un proiect asemănător cu multe altele deja existente?'; evident, ideea mea nu este să fac o copie a ceea ce există deja, *ideea este să învăţ din ideile bune şi din greşeliile altora* ca rezultatul final să se asemene doar ca şi nume, nu ca şi implementare.

Pe lângă problemele de securitate menţionate anterior, există şi alte probleme, vizibile clientului (greşeli ale programatorului), ce declasează site-ul, şi astfel numărul de accesări scade iar aplicaţia eşuează. Una dintre probleme: site-ul este *user-not-friendly* - clientul nu găseşte ceea ce caută, sunt 100 de butoane pe pagină dar niciunul nu face ce trebuie, sau este trimis de pe o pagină pe alta până ce se plictiseşte şi închide aplicaţia; unul din scopurile importante pe care le-am avut a fost să ofer clientului o navigare uşoară, gândindu-mă mereu că marea majoritate a utilizatorilor nu cunosc concepte de programare, deci totul trebuie să fie cât mai simplu cu putinţa. De asemenea *robustetea aplicaţiei* a fost încă un aspect important pe care l-am tratat cu maximă seriozitate; ştiind că un utilizator ai poate şi greşi am luat toate măsurile necesare ca aplicaţia să se comporte natural indiferent de cine şi cum o foloseşte.

În concluzie, am ales această temă deoarece sunt convinsă de utilitatea ei, este o aplicaţie care poate fi folosit de orice companie care doreşte să înceapă o afacere online; de asemenea, am încercat să depăşesc multe dintre problemele specifice acestui tip de aplicaţii, ca rezultatul să poată fi încadrat în cele mai înalte categorii.

## 1.3 Soluţii actuale

Aplicaţii web şi implicit aplicaţii ce simulează activitatea unui magazin pot fi realizate într-o mulţie de limbaje de programare; dintre acestea, amintim: *Java*, *c++*, *c#*, *PHP*, *Python* ca fiind cel mai des folosite[15], aşa cum se vede şi din tabelul de mai jos:

Website	Front-end(Client-side)	Back-End(Server-side)	Database
Google.com	JavaScript	C, C++, Go, Java, Python	BigTable
Facebook.com	Hack, PHP, C++, Java, Python, Erlang, D, Xhp	MySql, HBase	
YouTube.com	Flash, JavaScript	C/C++, Python, Java	MySql, BigTable
Wikipedia.org	JavaScript	PHP	MySql, MariaDB
Twitter.com	JavaScript	C++, Java, Scala, Ruby on Rails	MySql
Amazon.com	JavaScript	Java, C++, Perl	
eBay.com	JavaScript	Java	Oracle Database

Dintre limbajele de programare folosite pentru implementarea aplicaţiilor web se remarcă în mod special Java şi PHP;

Aceste 2 limbaje pot fi folosite pentru a îndeplini aceleaşi cerinţe; tocmai de aceea,este posibil să nu existe alte 2 limbaje care să presupună atâtea dez-bateri relativ la punctele forte, respectiv punctele slabe ale fiecăruia; ambele presupun aceeaşi acreditare şi au fost folosite în realizarea celor mai mari site-uri din lume;[17]

Ambele limbaje s-au născut în acelaşi an, 1995, Java dezvoltată de *Sun Microsystems* ca parte a platformei Java, iar PHP a fost creat special pentru web, ca un limbaj de scripting pe partea serverului pentru a fi încorporat în paginile HTML.[18]



Java	PHP
limbaj <i>strongly-typed</i> , adică este susţinut de un <i>compiler</i> ;dacă aşteptările compilerului nu sunt îndeplinite, programul nu va mai funcţiona până ce erorile nu sunt rezolvate;	limbaj <i>weakly-typed</i> , este mult mai flexibil, se bazează pe bunul simţ al programatorului; deşi sună mai atractiv pentru că presupune mai puţină cunoaştere, îngreunează îndeplinirea sarcinilor prin lipsa de standarde;
bine pregătit pentru modularizare; datorită instrumentelor suport ( <i>Ant/Maven, Javadoc, JUnit</i> ) framework-urile Java au artefacte mai uşor de instalat,mai bine documentate şi testate	modulele PHP expun în mod semnificativ mai multe probleme decât cele scrise în Java;unii dezvoltatori de module PHP inventează concepte proprii sau modulele sunt optimizate doar pentru un framework (ex: <i>Symfony</i> )
standardele definite permit o înţelegere facilă şi o mai mare eficienţă între membrii unei echipe(este mai bun pentru site-uri care folosesc module complexe)	mai accesibil programatorilor neexperimentaţi
JVM( <i>Java Virtual Machine</i> foarte mult optimizată)	performanţa scăzută(restartează maşina virtuală după fiecare request)
] limbaj open-source( <i>Sun</i> s-a angajat să îl facă open-source)	limbaj open-source,disponibil pe orice platformă
funcţionează universal, pe toate platformele;	funcţionează universal, pe toate platformele;
structurat pentru <i>programarea orientată pe obiecte</i>	necesitate de soluţii suplimentare pentru a putea programa orientat pe obiecte
foarte sigur	prezintă vulnerabilitaţi
clasele, metodele şi structurile sunt păstrate în memorie între 2 request-uri	clasele,metodele şi structurile sunt colectate de <i>garbage collector</i> (disponibil din 5.3) la sfârşitul unei cereri(se foloseşte în execuţie doar de ceea ce are nevoie);
există cadre( <i>framework</i> ), concepute pentru a crea aplicatii web	există cadre( <i>framework</i> ), concepute pentru a crea aplicatii web

# Capitol 2

## Aplicații web

### 2.1 Despre aplicații web

Un *web browser* este o aplicație soft ce permite user-ilor să obțină date și să interacționeze cu conținut localizat pe pagini web (ce aparțin unui *website*).

*Aplicațiile web* sunt programe ce permit vizitatorilor unui site să persiste și să obțină date dintr-o *bază de date* de pe internet folosind browser-ul preferat; aplicațiile web interoghează conținutul server-ului și generează în mod dinamic pagini web pe care le servește clientului; documentele sunt generate într-un format standard pentru a putea fi suportate de orice browser; un mare avantaj al acestui tip de aplicații este că *rulează indiferent de sistemul de operare și browserul folosit pe partea clientului*; de asemenea sunt foarte ușor de lansat ,fără costuri și fără instalări suplimentare;[14]

Avantajele folosirii aplicațiilor web:[16]

1. *ușor personalizabile*: interfața cu utilizatorul a aplicațiilor web este mai ușor de personalizat decât în cazul aplicațiilor desktop; acest lucru face mai ușoară personalizarea prezentației pentru diferite grupuri de utilizatori.
2. *accesibile pentru o gamă largă de device-uri*: conținutul poate fi văzut pe orice dispozitiv conectat la internet , inclusiv, PDA-uri, telefoane mobile ce extind abilitatea utilizatorului de a primi și de a interacționa cu informații.
3. *interoperabilitate îmbunătățită*: folosind tehnologii internet este posibilă o interoperabilitate mult mai bună între aplicații decât în cazul aplicațiilor desktop;

4. *instalare şi întreţinere*: instalarea şi întreţinerea devin mai puţin complicate; odată ce o nouă versiune sau îmbunătăţire este instalată pe serverul gazdă, toţi utilizatorii pot accesa imediat cum upgrade-urile sunt efectuate numai de către un profesionist cu experienţă la un singur server, rezultatele sunt mult mai previzibile şi de încredere.
5. *adaptabile la creşterea volumului de muncă*: dacă o aplicaţie necesită mai multă energie pentru a efectua sarcini, numai hardware-ul de server trebuie să fie modernizat; capacitatea software-ului poate fi îmbunătăţită rulând aplicaţia pe mai multe servere simultan; pe măsură ce creşte volumul de lucru, servere noi pot fi adăugate la sistem cu uşurinţă; dacă un server eşuează acesta poate fi înlocuit fără a afecta performanţa generală a aplicaţiei.
6. *tehnologii de bază flexibile*

## 2.2 Arhitectura client-server

Modelul client-server este o arhitectură distribuită care partajează task-uri între furnizori de servicii numiţi *servere* şi elemente care solicită servicii, numite *clienţi*. Clienţii şi serverele comunică printr-o reţea de calculatoare, de obicei prin Internet, având suporturi hardware diferite, dar pot rula şi pe acelaşi sistem fizic. Un server rulează unul sau mai multe programe server, care partajează resursele existente cu clienţii. Clientul nu partajează niciuna dintre resursele proprii, ci apelează resursele serverului prin funcţiile server. Clienţii iniţiază comunicaţia cu serverele şi aşteaptă mesajele acestora. Pentru menţinerea legăturii între cei doi, indiferent de pauzele care intervin, se foloseşte conceptul de sesiune, care de obicei este limitată în timp.

**Comunicarea client-server:** un serviciu este o abstracţie de resurse informatice şi un client nu trebuie să fie preocupat de modul în care serverul îndeplineşte cererea primită şi livrează răspunsul. Clientul trebuie doar să înţeleagă răspunsul bazat pe un *protocol bine cunoscut*;

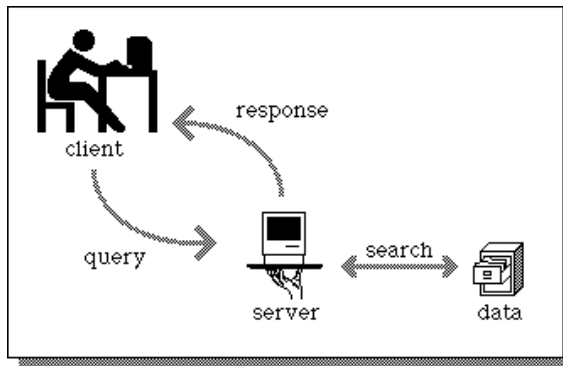


FIG. 2.1: arhitectura client-server

Clienţii şi serverele comunică bazându-se pe un model de mesagerie *cerere-răspuns*: clientul trimite o cerere, iar serverul returnează un răspuns. Pentru a comunica, calculatoarele trebuie să aibă un limbaj comun, şi trebuie să respecte normele, astfel încât atât clientul cât şi serverul să ştie la ce să se aştepte. Limba şi regulile de comunicare sunt definite într-un *protocol de comunicare*.

Un server poate primi cereri de la mai mulţi clienţi diferiţi, într-o perioadă foarte scurtă de timp. Deoarece computerul poate efectua un număr limitat de sarcini la un moment dat, el se bazează pe un sistem de planificare ce prioritizează cererile primite de la clienţi. Pentru a preveni abuzurile, software-ul server-ului limitează cum un client poate utiliza resursele serverului. Chiar şi aşa, un server nu este imun la abuzuri. Un atac exploatează obligaţia unui server de a procesa cererile prin bombardarea cu cereri fără încetare. Acest lucru împiedică capacitatea serverului de a răspunde la solicitările legitime.[23]

## 2.3 Protocolul HTTP

HTTP (*HyperText Transfer Protocol*) este protocolul folosit pentru schimbul de informaţii în Internet.

Datele se reprezintă utilizând în principal unul dintre următoarele limbaje:

- *eXtensible Markup Language (XML)* : subset al limbajului *Standard Generalized Markup Language (SGML)*
- *JavaScriptObjectNotation (JSON)* JSON

*HyperText Markup Language (HTML)*, *XHTML*, *HTML5* sunt principalele limbaje folosite pentru afişarea informaţiilor pe Web.[1]

### 2.3.1 Tranzacţie HTTP

Protocolul http este de tip client-server;

- Browser-ele *Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Opera, Apple Safari* sunt aplicaţii client uzuale;
- Informaţiile sunt găzduite / generate de servere Web de către site-uri şi servicii Web.[1]

**Atribuţiile clientului sunt următoarele:[1]**

1. *stabilirea conexiunii cu serverul web*
2. *trimiterea cererii către serverul web*
3. *recepţionarea răspunsului dat de serverul web*
4. *închiderea conexiunea*

Acest ciclu de acţiuni se numeşte *tranzacţie http*.

Serverul nu reţine informaţii între două tranzacţii http. Acest comportament se exprimă prin: **protocolul http este fără stare - stateless.**[1]  
Cererile şi răspunsurile sunt reprezentate ca linii de text separate de caracterul <CR><LF>, având structura:[1]

1. *preambul* -format dintr-o line
2. *antete* -0 sau mai multe attribute (nume:valoare)
3. *o linie goala*
4. *corpul mesajului* -facultativ

**Preambulul unei cereri** conţine:[1]

1. *numele metodei* (GET, POST, PUT, DELETE,etc)
2. *referinţa resursei: URL*
3. *versiunea protocolului http*

**Preambulul unui răspuns** conţine:[1]

1. *Versiunea protocolului http*
2. *codul răspunsului:*
3. *String explicitând codul răspunsului*

## 2.4 Cum funcţionează aplicaţiile web?

Figura alăturată prezintă cele 3 straturi ale unei aplicaţii web:[14]

1. *browser-ul web sau interfaţa clientului*
2. *instrumentul de generare al conţinutului*
3. *baza de date ce conţine datele clientului*

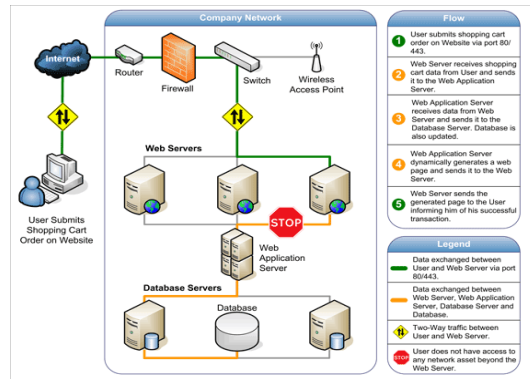
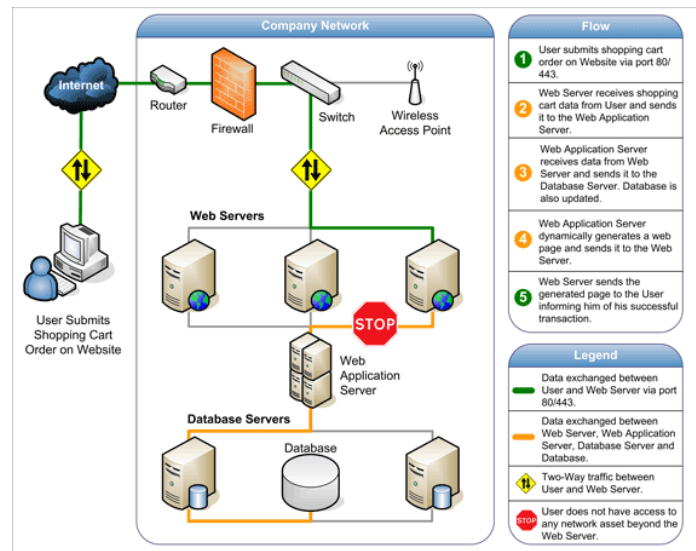


FIG. 2.2: structura unei aplicaţii web

Figura următoare prezintă cum cererea făcută de client este trimisă de browser prin internet serverului aplicaţiei; aplicaţia web accesează baza de date a serverului pentru a îndeplini cererea, găsind şi modificând conţinutul bazei; aplicaţia web prezintă apoi rezultatele user-ului prin browser;[14]



### Web security - probleme

În ciuda numeroaselor avantaje, aplicaţiile web ridică o serie de probleme legate de securitate care rezultă din programare necorespunzătoare; slăbiciuni serioase şi vulnerabilităţi permit *hacker-ilor* să câştige acces la bazele de date şi astfel să altereze datele pe care le găsesc;

Multe dintre aceste baze de date conţin *date valoroase (date personale, detalii legate de conturi bancare)*, devenind astfel o *ţintă frecventă pentru hackeri*;[14]

Unii hackeri, de exemplu, *injectează cod în aplicaţiile web vulnerabile* pentru a păcăli utilizatorii şi pentru a-i redirecţiona pe aşa numitele *phishing sytes* -> *site-uri de piraterie*; această tehnică este numită **Cross-site scripting** şi este folosită chiar dacă aplicaţia este foarte robustă;

Ultimele cercetări arată ca 75% din atacuri au loc la nivelul aplicaţiei web;[14]

- *Firewall-urile şi SSL (Secure Sockets Layer)* nu oferă protecţie împotriva hacker-ilor, pur şi simplu pentru că accesul la site trebuie să fie făcut public. Toate sistemele de baze de date moderne pot fi accesate prin porturi specifice (ex.: portul 80 şi 443) şi oricine poate încerca conexiuni directe la bazele de date, *ocolind în mod eficient mecanismele de securitate folosite de sistemul de operare*. Aceste porturi rămân deschise pentru a permite comunicarea cu trafic legitim şi, prin urmare, constituie o vulnerabilitate majoră.
- Aplicaţiile web au adeseori acces la *backend data*, cum ar fi bazele de date şi prin urmare au control asupra datelor valoroase care sunt foarte greu de securizat; cei care nu au acces direct la bază au la îndemână un script cu ajutorul căruia transmit şi primesc date; dacă un hacker află de vreo vulnerabilitate într-un astfel de script poate accesa anumite date, şi o dată ce le obţine le poate folosi după bunul plac;
- Multe aplicaţii web sunt făcute "la comandă", şi implică un grad mai mic de testare decât cele în conformitate cu un format standardizat; în consecinţă, aplicaţiile custom sunt mai susceptibile la atacuri;[14]

## Capitol 3

# Unelte de dezvoltare

### 3.1 Sistem de versionare.Tortoise SVN

Fiind o aplicație complexă, persistarea schimbărilor apărute pe parcursul dezvoltării a fost imperios necesară; pentru aceasta și pentru a nu păstra o mulțime de copii ale proiectului ce ar fi ocupat mult spațiu, am folosit un sistem de control al versionării.

Avantajele folosirii unui sistem de versionare:[22]

1. *salvări atomice*: fie se salvează toate modificările, fie nicuna; acest lucru permite dezvoltatorilor să privească modificările ca pe niște entități logice;
2. *versionări ale directoarelor*: urmărește modificările aduse întregilor arborilor de directoare de-a lungul timpului;
3. *ramificare și etichetare eficientă*: costul de ramificare și etichetare nu trebuie să fie proporțional cu dimensiunea proiectului; sistemele de versionare creează ramuri și tag-uri prin simpla copiere a proiectului, folosind un mecanism similar cu un hard-link; astfel, aceste operațiuni durează doar o cantitate foarte mică, constantă de timp, și foarte puțin spațiu în depozit.
4. *administrare bună a datelor*: sistemele calculează diferențe de fișiere folosind un algoritm de diferențiere binar, care funcționează identic în fișiere text și binare; ambele tipuri de fișiere sunt stocate în mod egal în depozit, iar diferențele sunt transmise în ambele direcții în întreaga rețea.



### Tortoise SVN

Tortoise SVN este un program gratuit, creat pentru a ajuta programatorii să administreze diferite versiuni ale codului sursă; programul a fost creat în anul 2002, în limbajul c++ şi funcţionează doar pe sistemul de operare *Microsoft Windows*[22]



TortoiseSVN

TortoiseSVN gestionează fişiere şi directoare de-a lungul timpului. Fişierele sunt stocate într-un depozit central. Depozitul este asemănător cu un server de fişiere obişnuit, cu excepţia faptului că îşi aminteşte fiecare modificare făcută vreodată fişierelor şi directoarelor. Acest lucru vă permite să recuperaţi versiunile mai vechi ale fişierelor şi să examinaţi cine, ce, şi când a schimbat. Acesta este motivul pentru mulţi oameni cred despre sistemele de subversionare şi de control al versiunii, în general, că sunt un fel de *maşină a timpului*. [22]

Avantajele folosirii sistemului Tortoise SVN:[22]

1. *integrare totală*: se integrează perfect în mediul shell Windows; acest lucru înseamnă că că puteţi continua munca cu *tool-urile* cu care snteţi deja familiarizaţi; de asemenea nu este obligatorie utilizarea Windows Explorer; meniurile TortoiseSVN funcţionează si cu alţi manageri de fişiere. ar trebui, totuşi, să se aibă în vedere faptul că TortoiseSVN este dezvoltat în mod intenţionat ca o extensie pentru Windows Explorer;
2. *pictograme*: statusul fiecărui fişier sau director versionat este indicat de mici pictograme; astfel, poţi vedea care este statusul copiei pe care lucrezi;
3. *interfaţă grafică*: când listezi schimbările unui anumit fişier, poţi vedea comentariile acelei schimbări; de asemenea, poţi vedea lista fişierelor modificate;
4. *acces uşor la comenzi*: toate comenzile sunt disponibile din meniul contextual Explorer; tortoise adaugă propriul său submeniu acolo.

## 3.2 Maven

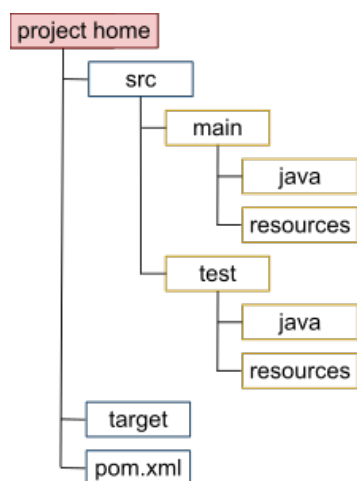
Maven a fost iniţial conceput ca o încercare de a simplifica build-ul proceselor în cadrul proiectului Jakarta Turbine; se căuta o modalitate standard de a realiza build-ul unui proiect, o definiţie clară a componentelor unui proiect, un mod uşor de a publica informaţii despre un anumit proiect şi un mod de a împărtăşi fişiere JAR;

Rezultatul este un tool ce poate fi folosit pentru build-ul şi administrarea proiectelor bazate pe Java; acesta este dotat cu obiective predefinite pentru realizarea unor sarcini bine definite cum ar fi *compilarea şi împachetarea codului*; descarcă automat librării Java şi plugin-uri Maven din unul sau mai multe depozite cum ar fi *Maven 2 Central Repository* şi le depune într-un cache local; această zonă locală poate fi actualizată cu artefacte create în proiecte locale;

Mecanismul de manipulare a dependenţelor Maven este organizat în jurul unui sistem de coordonate ce identifică artefactele. Un proiect care are nevoie de, să zicem, *Hibernate*, trebuie pur şi simplu să declare coordonatele bibliotecii în fişierul pom.xml. Maven va descărca în mod automat dependenţa şi dependenţele de care Hibernate are nevoie (*dependinţe tranzitive*) şi le va stoca în depozitul local al utilizatorului. *Maven 2 Central Repository* este folosit în mod implicit pentru a căuta biblioteci, dar utilizatorul poate configura depozitele care vor fi utilizate, în cadrul POM;[26]

### Obiective Maven

- *realizare uşoară a build-ului* - elimină detalii nefolositoare
- *sistem uniform de build-uire* - permite build-uirea unui proiect, folosind fişierul POM şi un set de plugin-uri, care sunt împărtăşite de toate proiectele care utilizează Maven, oferind un sistem uniform pentru build
- *furnizare de informaţii de calitate despre proiect* - oferă informaţii folosite preluat de obicei din fişierul POM şi generate de sursele proiectului (liste cu dependenţe, rezultatele unit-testing-ului, surse referenţiate);
- *migrare transparentă către noi caracteristici* - furnizează clienţilor o modalitate uşoară de a-şi actualiza instalările pentru a profita de schimbările suferite de Maven;
- *orientare spre bune practici de dezvoltare* - adună principii pentru dezvoltarea celor mai bune proiecte;[26]



Numele directorului	Scop
project home	fişierul pom.xml şi toate subdirectoarele
src/main/java	codul sursă livrabil al proiectului Java
src/main/resources	resursele livrabile ale proiectului, cum ar fi fişierele de proprietăţi
src/test/java	codul sursă pentru testare
src/test/resources	resursele necesare testării

Table 3.1: Structura directoarelor unui proiect Maven

Proiectele Maven sunt configurate folosind *Project Object Model*, ce este salvat în fişierul *pom.xml*; Acest POM defineşte doar un identificator unic al proiectului (coordonate) şi al dependenţelor de framework-ul JUnit; acesta este suficient pentru construirea proiectului şi rularea unit-testelor asociate cu proiectul;

Folosind Maven, utilizatorul doar configurează proiectul, în timp ce plugin-urile configurate fac efectiv lucrul de elaborare a proiectului, de curăţare a directoarelor target, de rulare a unit-testelor, de generare a documentaţiei şi aşa mai departe.

# Capitol 4

## Persistarea datelor

### 4.1 Persistarea datelor în baze de date

Persistența este mecanismul utilizat de aplicații pentru a păstra datele, date ce altfel ar fi pierdute la închiderea aplicației sau a calculatorului, într-un context persistent, precum o bază de date. Persistența datelor este o necesitate în cadrul proiectelor mari, așa cum este și aplicația de față; între a salva datele într-un fișier sau într-o bază de date, am ales baza de date datorită următoarelor avantaje pe care le aduce:[27]

1. *suportă interogări*, deci programatorul nu trebuie să caute manual prin fișier;
2. *suportă interogări bazate pe indecsi*, deci este foarte rapidă;
3. *suportă acces concurențial*
4. *suportă acces din rețea*
5. *are grijă de integritatea datelor*
6. *poate actualiza datele ușor*
7. *sunt sigure*
8. *ușor de manipulat din punctul de vedere al programatorului*

### 4.1.1 MySql

Unul din cele mai raspândite sisteme de gestiune a bazelor de date este MySQL.

La început, MySQL era un SGBD care ducea lipsă de facilităţi precum suportul pentru tranzacţii sau procedurile stocate, dar în ultima vreme, acestea şi unele facilităţi noi au fost incluse în distribuţiile MySQL, astfel încât acesta a devenit un SGBD foarte complex, intrând în categoria enterprise.

Dintre toate SGBD-urile existente în prezent (SQL server, MySql, Oracle, etc) am ales să folosesc MySql din următoarele motive:[3]

1. *Prezintă performanţe foarte bune dar este mult mai uşor de administrat şi folosit decât alte SGBD-uri.*
2. *MySQL foloseşte SQL (Structured Query Language), care este cel mai folosit standard în piaţă.*
3. *Are numeroase capabilităţi este multi-threading (ceea ce înseamnă că mai mulţi clienţi se pot conecta simultan), poate fi folosit în mai multe moduri (prin aplicaţii client, consolă, aplicaţii client grafice, aplicaţii client web, prin diverse interfeţe de programare de către limbaje precum C, C++, Java, PHP etc.).*
4. *Sigur: include straturi de securitate ce protejează datele sensibile de intruşi*
5. *MySQL este unul din cele mai rapide SGBD-uri de pe piaţa.:designerii MySQL au luat decizia de a oferi mai puţine caracteristici decât alţi concurenţi de baze de date importante,având ca prim interes viteza;*
6. *Este un program open-source, care în cele mai multe cazuri poate fi folosit free.*
7. *Poate fi folosit în reţea şi chiar accesat de pe Internet;prezintă posibilitatea de a lucra cu el folosind SSL (Secure Sockets Layer).*
8. *Este foarte portabil*
9. *Are o dimensiune foarte mică*
10. *Administrează bine memoria: a fost complet testat pentru a preveni memory-leaks*
11. *Scalabil: poate manevra un volum mare de date,pâna la 50 milioane de înregistrări*

12. *Rulează pe mai multe sisteme de operare*
13. *Este folosit pe scară largă, ceea ce face ca documentaţia să fie foarte uşor de obţinut.*

## Arhitectura MySql

MySQL are o arhitectură de tip client/ server, ceea ce înseamnă că există un program principal care rulează, şi mai multe programe (programe client) care se conectează la el, fac cereri, primesc răspunsuri;

Când instalăm MySQL, instalăm următoarele:[3]

1. **Serverul MySql:mysql:** aplicaţia care răspunde cererilor şi face managementul accesului la disk sau la memorie; este *multi-threaded*, adică suportă mai mulţi clienţi simultan; serverul rulează în fundal (*daemon*); invocarea "mysqld" porneşte serverul MySql pe sistem, iar încheierea aplicaţiei va opri serverul;
2. **Aplicaţii client:** aplicaţii gândite pentru ca noi să putem accesa, prelucra datele din bazele de date; noi folosim aceste programe care se conectează la server pentru fiecare din operaţiile pe care le facem; există mai multe tipuri de astfel de aplicaţii:
  - (a) *MySQL Workbench:* un instrument vizual unificat pentru arhitecţi şi dezvoltatori de baze de date; oferă facilităţi pentru modelarea datelor, dezvoltare SQL, instrumente complete de administrare şi de configurare a serverului, administrarea utilizatorilor, de backup, etc; oferă de asemenea instrumente vizuale pentru crearea, executarea, precum şi optimizarea interogărilor SQL; editorul oferă evidenţierea sintaxei cu ajutorul culorilor, auto-complete, reutilizarea fragmentelor SQL, şi istoricul de executare a interogărilor SQL.[46]
  - (b) *mysql:* aplicaţie în linie de comandă care permite transmiterea interogărilor şi vizualizarea răspunsurilor;

## 4.2 JPA

JPA este un framework lightweight ce foloseşte POJO (*Plain Old Java Objects*) pentru a persista obiecte ce reprezintă date relaţionale. JPA stabileşte relaţionări ale modelului persistent, prin definirea componentelor entitate. API-ul defineşte clasa entitate ca un echivalent al unei tabele din baza de date, pe partea de business a aplicaţiei. O instanţă a entităţii este definită ca un obiect, echivalent al unei linii din tabela bazei de date. JPA furnizează programatorilor Java facilitatea de mapare obiect/relatie, pentru a gestiona modelul relaţional al bazelor de date implicate în aplicaţiile Java.[30]

**Beneficiile utilizării JPA:**[30]

1. nu trebuie să creăm obiecte complexe de acces la date (DAO)
2. API-ul este folosit pentru a gestiona tranzacţii
3. codul de interacţiune cu baza de date este standard, indiferent de vendorul bazei de date relaţionale
4. putem evita SQL şi în schimb putem folosi un query language, orientat pe obiect
5. putem folosi JPA pentru persistenţa aplicaţiilor desktop

**Concepte fundamentale:**[30]

- *Entitatea*, este utilizată pentru a reprezenta un tabel relaţional într-un obiect Java
- *Unitate de persistenţă*, defineşte mulţimea tuturor claselor ce au legătură cu aplicaţia şi care sunt mapate unei singure baze de date; unităţile de persistenţă sunt definite de fisierul de configurare *persistence.xml*
- *Context persistent*, este o mulţime de instanţe de entităţi în care este o unică instanţă a entităţii pentru orice identificator de entitate persistentă
- *Entity manager*, face munca de creare, citire şi scriere a entităţilor

### 4.2.1 Entităţi

Entitatea reprezintă o structură de date prin intermediul căreia se asigură persistenţa. O entitate este asociată unui tabel al unei baze de date relaţionale, iar fiecare instanţă a entităţii corespunde unei linii din tabelă. Câmpul unei clase entitate corespunde unei coloane din tabela bazei de date.[30]

**Cerinţe pentru clase entităţi:[4]**

1. clasa trebuie să fie adnotată cu adnotarea *javax.persistence.Entity*
2. clasa trebuie să aibă un constructor implicit public sau protected; mai poate avea şi alţi constructori
3. clasa nu poate fi declarată *finală*;
4. dacă o instanţă a clasei este trimisă ca parametru al unei metode, precum în cazul interfeţelor remote session bean, clasa trebuie să implementeze interfaţa *Serializable*
5. poate extinde clase entitate cât şi clase non-entitate şi clasele non-entitate pot extinde clase entitate
6. câmpurile unei clase entitate pot fi declarate doar private, protected şi pot fi accesate doar prin intermediul metodelor publice ale clasei
7. clasele entitate nu pot defini metoda *finalize()*.

**Ciclul de viaţă al unei entităţi:[4]**

1. *New* -se obţine în urma folosirii operatorului *new*; în acest caz nu este nicio linie din tabelă în corespondenţă cu cheia primară a entităţii, din nivelul persistent asociat;
2. *Managed* -există o linie în corespondenţă şi datele sunt ținute sincronizate, de către persistence provider, cu datele din entitate
3. *Detached* -la fel ca mai sus doar că datele din entitate nu sunt ținute sincronizate
4. *Removed* -reprezintă o ştergere în aşteptare a datelor, din linia corespondentă din baza de date



### Chei primare în entităţi[4]

Fiecare entitate are un obiect unic pe post de identificator, cunoscut sub numele de cheie primară. Cheia primară poate fi simplă sau compusă. Cheile primare simple utilizează anotaţia *javax.persistence.Id* pentru a marca câmpul. Cheile primare compuse (formate din mai multe attribute), trebuie definite într-o clasă cheie primară. Cheile primare compuse sunt notate folosind anotaţiile *javax.persistence.EmbeddedId* şi *javax.persistence.IdClass*.

## 4.2.2 Relaţionarea entităţilor

Următoarele patru proprietăţi sunt folosite pentru a descrie asocierea între obiecte:[30]

1. *Cardinalitatea*: specifică numărul de relaţionări între două entităţi relaţionate, poate fi de următoarele tipuri: *one-to-one*, *one-to-many*, *many-to-one*, *many-to-many*
2. *Direcţia*: care determină navigabilitatea şi vizibilitatea, poate fi:
  - **unidirecţională**: doar o singură entitate dintre cele două implicate în relaţie poate vedea cealaltă entitate
  - **bidirecţională**: fiecare entitate poate vedea cealaltă entitate din relaţie; putem astfel include cod în oricare dintre entităţi pentru a naviga către cealaltă entitate în vederea obţinerii de informaţii şi servicii de la aceasta.
3. *Proprietarul relaţiei (owner)*: specifică partea de *owning* a relaţionării, care la rândul său conţine maparea fizică; cealaltă parte a relaţiei se numeşte *inverse side*.
4. *Tipul de propagare al relaţiei*: se referă la propagarea efectului unei operaţii entităţilor asociate; se exprimă prin termenii: *All*, *Persist*, *Merge*, *Remove*, *Refresh*, *None*

## 4.3 Hibernate

Hibernate este o soluţie Object-Relational Mapping (ORM) pentru JAVA construită să medieze interacţiunile dintre o aplicaţie şi o bază de date relaţională.

Hibernate este o implementare a specificaţiilor JPA (cel mai popular furnizor). Ca atare, poate fi utilizat cu uşurinţă în orice mediu ce suportă JPA, inclusiv aplicaţii Java SE, Java EE.

Atunci când există noi modificări în cadrul specificaţiilor JPA, Hibernate eliberează o variantă actualizată a implementării acelor specificaţii.

Hibernate permite programatorului să dezvolte clase persistente urmărind principiile programării orientate pe obiect inclusiv moştenire, polimorfism, asociere, compoziţie;

Maapează clase Java la tabele SQL şi tipurile de date din Java la cele din SQL, eliberând astfel programatorul de 95% din taskurile aferente persistării;[29] oferă, de asemenea facilităţi de interogare şi recuperare a datelor din bază; generează automat apeluri SQL şi scuteşte dezvoltatorul de manipularea seturilor de date obţinute şi de conversia obiectelor;

Arhitectura Hibernate este astfel stratificată pentru a menţine programatorul izolat de detaliile specificaţiilor JPA. Hibernate foloseşte baze de date şi datele configurate pentru a furniza servicii de persistenţă la cerere.

Maparea claselor Java la tabele SQL se realizează prin configurarea unui *fişier XML* sau prin folosirea *adnotărilor Java*; când se folosesc fişiere xml, Hibernate oferă cod sursă schelet pentru clasele persistate;[2]

De asemenea, sunt asigurate facilităţi de aranjare a relaţiilor *one-to-many*, *many-to-many*; în plus faţa de gestionarea relaţiilor între obiecte, Hibernate poate administra *relaţii reflexive*, unde un obiect are relaţii cu alte instanţe ale aceluiaşi tip;

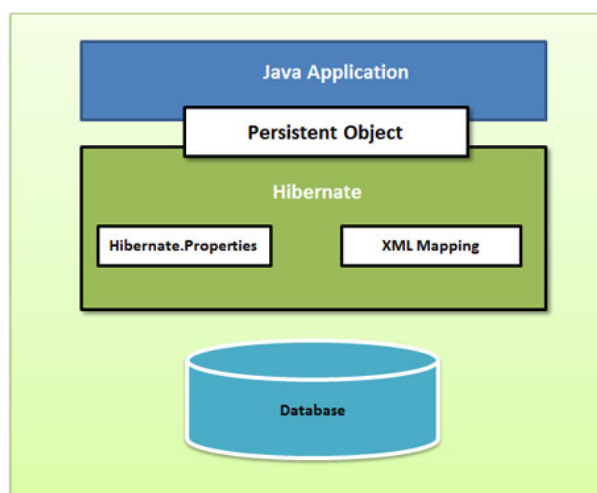


FIG. 4.1: hibernate

## Capitol 5

# Web server vs Application server

Un *web server* manipulează doar cereri bazate pe protocolul HTTP, pe când un *server de aplicație* servește partea de bussines-logic aplicației prin orice număr de protocoale.

### **Web server:**

Când un server web primește o cerere HTTP, răspunde cu un răspuns HTTP, cum ar fi o pagină HTML; pentru a procesa o cerere, un server Web poate răspunde cu o pagină HTML statică sau cu o imagine, poate trimite o redirecționare, sau delega răspunsul unui alt program. Modelul de delegare al web server-ului este destul de simplu; când o cerere ajunge la server, serverul pur și simplu trimite cererea programului cel mai capabil să o rezolve; serverul nu oferă nici o altă funcționalitate în afară de a oferi un mediu în care programul să se execute și să trimită înapoi răspunsul generat;

### **Application server:**

Un server de aplicații expune business logic aplicațiilor client prin diverse protocoale, inclusiv HTTP. În timp ce un server Web se ocupă în principal cu trimiterea paginilor HTML pentru afișare într-un browser Web, un server de aplicații oferă acces la logica de afaceri pentru a fi utilizată de către aplicații client. Aplicația clientului poate folosi aceste funcționalități la fel cum ar cheta o metodă a unui clase.[24]

Să luăm ca și exemplu aplicația de față, un magazin on-line ce oferă informații despre prețurile produselor în timp real; când clientul alege un produs, site-ul efectuează o căutare și întoarce rezultatele încorporate într-o pagină HTML; Site-ul poate implementa această funcționalitate în mai multe feluri; voi prezenta 2 scenarii: unul ce utilizează un server de aplicație, și unul ce nu folosește un server de aplicație;

1. **Web server fără server de aplicație:** în acest scenariu doar un web server oferă funcționalitate site-ului; serverul preia cererea clientului,

iar apoi o trimite programului capabil să o rezolve; programul caută preţul produsului în baza de date; când a obţinut rezultatul, programul foloseşte rezultatul pentru a formula pagina HTML, pe care serverul web o va trimite browserului;

2. **Web server cu server de aplicaţie:** este asemănător cu primul scenariu, în sensul că tot web serverul delegă răspunsurile generate; cu toate acestea, se poate pune acum logica de business pentru căutarea preţului pe un server de aplicaţii; astfel, în loc ca script-ul să ştie cum să caute în baza de date şi să formuleze un răspuns, script-ul poate apela pur şi simplu serviciul de căutare al serverului de aplicaţie. Separând partea de logică de codul generator de pagini HTML, logica devine mult mai reutilizabilă între aplicaţii; un al doilea client, ar putea folosi acelaşi serviciu; în primul scenariu, serviciul de căutare al preţului nu este reutilizabil pentru că informaţia este încorporată în pagina HTML; în concluzie, în scenariul al doilea web serverul se ocupă de cererile HTTP, răspunzând cu pagini HTML, în timp ce serverul de aplicaţie oferă partea de business logic prin serviciul de căutare al preţului;

## 5.1 JBoss 7.1.1 AS

Luând în considerare avantajele aduse de folosirea unui server de aplicaţie, am ales un astfel de server în aplicaţia mea, şi anume **JBoss 7.1.1 AS**.

Printre avantajele folosirii serverului de aplicaţie JBoss se numără şi următoarele:

- *performanţa ridicată* -oferă viteză nemaipomenită; serviciile serverului se întâmplă concurent pentru a elimina orice timp de aşteptare;
- *testare uşoară*
- *administrare elegantă*- centralizată pe client
- *teste interne* -ce asigură respectarea regulilor
- *manevrare bună a memoriei*
- *design modular* -ce permite ca modulele serverului să ofere o bună izolare a aplicaţiei încărcând clasele de care aplicaţia are nevoie şi ascunzând clasele implementate de server
- *configurare simplă*

# Capitol 6

## View

### 6.1 JSF

JSF -*Java Server Faces* este un framework web bazat pe pattern-ul MVC folosit pentru a simplifica construirea interfețelor client în cadrul aplicațiilor web prin reutilizarea componentelor UI; oferă facilitatea de a lega widget-uri din interfața grafică de event-handle-uri de pe partea serverului; specificațiile JSF definesc un set standard de componente UI și oferă un API(Application programming Interface) pentru dezvoltare de componente;[47]

#### 6.1.1 Beneficii JSF

JSF reduce din efortul în creerea și menținerea aplicațiilor ce rulează pe un server și restituie componente grafice unui client; facilitează aplicațiile web prin:[5]

1. *oferirea de componente grafice reutilizabile*
2. *facilitarea transferului de date între componente grafice*
3. *gestionarea stării UI între mai multe cereri ale serverului*
4. *permiterea implementării de componente personalizate*
5. *legarea evenimentelor de pe partea clientului de cod scris pe partea serverului*

Tehnologia JSF este un framework folosit pentru dezvoltarea , construirea și folosirea interfețelor grafice în cadrul unei aplicații web; tehnologia este bazată pe design-pattern-ul MVC(*Model View Controller*)pentru a separa prezentarea de logică;

### 6.1.2 Arhitectura JSF

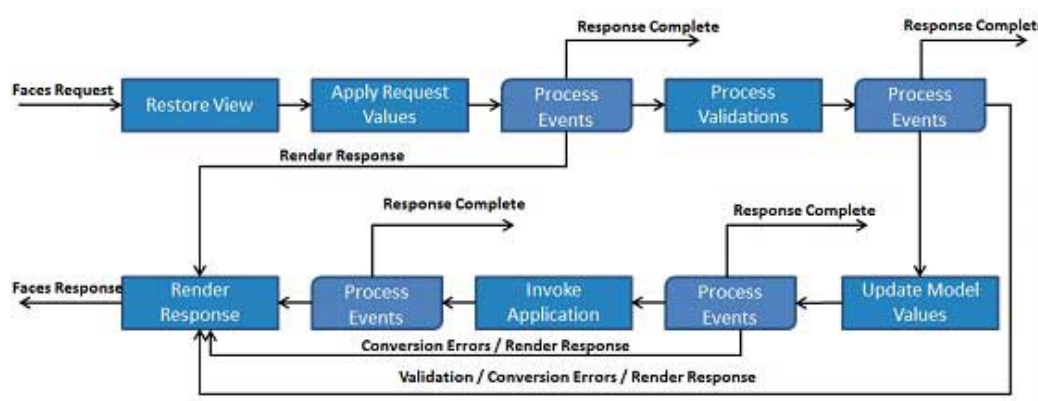
O aplicaţie JSF este similară cu o aplicaţie Java web obişnuită; rulează într-un server web şi conţine:[5]

1. *componente JavaBean care conţi funcţionalităţi specifice aplicaţiei*
2. *biblioteci custom*
3. *componente UI reprezentate ca obiecte stateful pe partea serverului*
4. *clase ajutătoare pe partea serverului*
5. *fişiere de configurare a resurselor*

### 6.1.3 Ciclul de viaţă al unei aplicaţii JSF

Ciclul de viaţă al unei aplicaţii JSF constă din 6 faze, şi anume:[5]

- *restaurarea interfeţei grafice*
- *trimiterea cererii*
- *validarea proceselor*
- *actualizarea valorilor modelului*
- *trimiterea răspunsului*



Cele 6 faze arată ordinea în care JSF procesează un formular; figura de mai sus arată fazele în ordinea probabilă de execuţie;

## 6.2 Managed Beans

O clasă Managed Bean este o clasă Java Bean obișnuită înregistrată cu JSF; cu alte cuvinte, este un bean administrat de framework-ul JSF; Clasele Managed Bean conțin *accesori*, *modificatori* și *business-logic*; aceste metode pot fi apelate din fișiere jsf;[47]

În JSF 1.2 clasele Managed Bean trebuiau configurate în fișierul *faces-config.xml*; din JSF 2.0 bean-urile pot fi ușor înregistrate folosind *adnotări*;

### 6.2.1 Adnotări

#### Adnotarea @Managed Bean

Definește un managed bean cu numele specificat în atributul *name*; dacă atributul nu este specificat, numele va fi chiar numele clasei.

Un alt atribut important este *eager*; dacă este setat **true**, atunci bean-ul este creat înainte să fie cerut prima dată; în caz contrar, va fi creat doar când este cerut;[47]

#### Adnotările de tip scop

Acest tip de adnotări setează scopul în care va fi plasat managed bean-ul; dacă scopul nu este precizat, va fi setat implicit la *request scope*;[47]

Scop	Descriere
@RequestScoped	Bean-ul trăiește cât timp trăiește tranzacția HTTP; este creat la o cerere HTTP și este distrus când răspunsul asociat este trimis;
@NoneScoped	Bean-ul trăiește cât o singură evaluare EL( <i>Expression Language</i> ); este distrus imediat după evaluare;
@ViewScoped	Trăiește atâta timp cât utilizatorul interacționează cu aceeași pagină JSF în browser; este creat la o cerere HTTP și se distruge când utilizatorul schimbă pagina;
@SessionScoped	Trăiește cât durata de viață a sesiunii HTTP; este creat la prima cerere ce implică bean-ul în sesiune și este distrus când sesiunea este invalidată
@ApplicationScoped	Durata de viață a bean-ului este durata de viață a aplicației ;este creat la prima cerere ce implică bean-ul în aplicație sau când aplicația pornește și atributul <i>eager</i> este setat true și este distrus când aplicația este oprită
@CustomScoped	se distruge după o anumită perioadă de timp în care este inactiv, adică instanța bean-ului nu este folosită

### Adnotarea @ManagedProperty

JSF este un framework bazat pe injecţie statică de dependenţe; folosind adnotarea un managed bean poate fi injectat într-un alt bean;[47]

## 6.3 Primefaces

*PrimeFaces* este o bibliotecă open-source pentru JSF 2.0 ce conţine peste 100 de componente; este mult mai bună decât multe alte biblioteci pentru JSF din următoarele motive:[6]

- *colecţie bogată de componente UI*: HtmlEditor, Dialog, AutoComplete, Charts şi multe altele
- *nu necesită configurări sau dependenţe extra*
- *documentaţie extinsă cu exemple*
- *AJAX integrat*
- *peste 35 de teme*
- *suport de la Atmosphere Framework*
- *kit Mobile UI pentru crearea de aplicaţii pe dispozitive portabile*

PrimeFaces are un singur jar, **primefaces-version.jar**; sunt 2 modalităţi de a descărca acest jar, ori de pe pagina gazdă PrimeFaces, ori se poate declara ca dependenţă:[6]

1. descărcare manuală: 3 artefacte diferite sunt disponibile pentru fiecare versiune PrimeFaces: *binar*, *sursă* şi *pachet*; artefactele sunt disponibile la următoarea adresă: <http://www.primefaces.org/downloads.html>
2. descărcare prin Maven: id-ul grupului este *org.primefaces*, iar id-ul artefactului este *primefaces*; dependenţele se adaugă în fişierul *pom.xml* al proiectului maven creat:[6]

```
<dependency>
<groupId>org.primefaces</groupId>
<artifactId>primefaces</artifactId>
<version>4.0</version>
</dependency>
```



### 6.3.1 XHTML

XHTML vine de la *eXtensible HyperText Markup Language* şi reprezintă trecerea de la HTML la XML; a fost creat din 2 mari motive:

- pentru a crea un standard mai strict pentru crearea paginilor web, reducând incompatibilităţi între browsere
- pentru a crea un standard care poate fi folosit pe o varietate de dispozitive diferite fără modificări

Cel mai important lucru despre XHTML, este faptul că este aproape la fel ca HTML, deşi este mult mai important ca şi codul să fie scris corect; este imposibil ca un cod prost scris să fie compatibil cu XHTML; spre deosebire de HTML, unde simple erori (cum ar fi lipsa unei etichete de închidere) sunt ignorate de către browser, codul XHTML trebuie să respecte întocmai specificaţiile; acest lucru se datorează faptului că browser-ele în dispozitive portabile, nu au puterea de a afişa pagini prost formate, astfel XHTML asigură corectitudinea codului, astfel încât acesta să poată fi utilizat pe orice tip de browser.[31]

**Diferenţe faţă de HTML:[31]**

- *tag-urile trebuie scrise cu litere mici*
- *toate documentele trebuie să aibă doctype*
- *toate documentele trebuie să respecte formatul*
- *toate tag-urile trebuie închise*
- *atributele trebuie adăugate în mod corespunzător*
- *atributele nu pot fi scurtate*
- *tag-urile trebuie să fie corect imbricate*

### 6.3.2 AJAX

AJAX vine de la *Asynchronous JavaScript and XML*; este o nouă tehnică de a crea aplicaţii web mai bune, rapide şi interactive folosind XML, HTML, CSS şi JavaScript.

Foloseşte XHTML pentru conţinut, CSS pentru prezentare şi JavaScript pentru afişare dinamică de conţinut;

Aplicaţiile web convenţionale transmit informaţii la şi de la server folosind cereri sincrone; acest lucru înseamnă că utilizatorul completează un formular, execută şi este redirectat spre o pagină care conţine răspunsul; cu Ajax, când butonul *submit* este apăsă, JavaScript face o cerere către server, interpretează răspunsul şi actualizează pagina, astfel, utilizatorul nici nu va şti că ceva a fost trimis serverului;

Un utilizator poate continua să folosească aplicaţia în timp ce programul client face cereri către server; utilizând AJAX, nici nu este nevoie de click pentru a declanşa un eveniment, o simplă mişcare din mouse este suficientă; [33]

### 6.3.3 JavaScript

JavaScript este un limbaj de programare interpretat cu capabilităţi orientate pe obiect, care permite interactivitate în paginile statice HTML;

Script-urile trebuie să fie incluse sau referite de un document HTML, astfel codul va fi interpretat de către browser; Aceasta înseamnă că o pagină web nu mai este nevoie să conţină HTML static, poate include programe care interacţionează cu utilizatorul, controlează browser-ul, şi crează în mod dinamic conţinut HTML. De exemplu, se poate utiliza JavaScript pentru a verifica dacă utilizatorul a introdus o adresă de e-mail validă într-un câmp al unui formular.

Codul JavaScript este executat atunci când utilizatorul trimite formularul, şi numai dacă toate intrările sunt valabile vor fi trimise serverului.

JavaScript poate fi folosit pentru a capta evenimente iniţiate de utilizator precum clickurile butoanelor de navigare, precum şi alte acţiuni pe care utilizatorul le iniţiază în mod explicit sau implicit.[34]

#### **Avantajele JavaScript[34]**

1. *interacţiune mai puţină cu serverul:* datele introduse de utilizator se pot valida înainte de a fi trimise la server; acest lucru economiseşte traficul, ceea ce înseamnă mai puţin timp de încărcare
2. *feedback rapid utilizatorului:* utilizatorul nu trebuie săştepte ca pagina să se reîncarce pentru a verifica dacă uitat vreun câmp necompletat
3. *interactivitate crescută:* se pot crea interfeţe ce reacţionează dacă utilizatorul le activează prin intermediu mouse-ului sau al tastaturii
4. *interfeţe bogate:* se pot include elemente de drag-and-drop sau slidere pentru a oferi o interfaţă ogată vizitatorilor site-ului

## 6.4 CSS

CSS sau *Cascading Styles Sheets*, este o modalitate de a stila şi prezenta paginile HTML. În timp ce HTML reprezintă conţinutul, stilul este prezentarea acestui document.

Stilurile nu se aseamănă deloc cu limbajul HTML, ele au un format: *proprietate: valoare*, iar cele mai multe proprietăţi pot fi aplicate pentru cele mai multe tag-uri HTML.[32]

Există 3 feluri de a aplica CSS unei pagini HTML:

1. **in-line**: inserate în tag-urile HTML folosind atributul *style*

```
<p style="color: red">text</p>
```

2. **încadrate**: valabile pentru toată pagina:

```
<!DOCTYPE html>
<html>
<head>
<title>CSS Example</title>
<style>
  p {
    color: red;
  }
</style>
```

3. **externe**: se scriu & în fişiere separate:

```
p {
  color: red;
}

a {
  color: blue;
}

<!DOCTYPE html>
<html>
<head>
  <title>CSS Example</title>
  <link rel="stylesheet" href="style.css">
```

# Capitol 7

## Controller

### 7.1 EJB

*Enterprise Java Bean* este un framework care înglobează pe o serie de implementări ale interfețelor de programare resurse care sunt utilizate prin intermediul unei componente Enterprise Java Bean- EJB. O componentă EJB este o clasă Java care face parte din aplicația server, conține metodele care rezolvă o problemă și este conținută într-un server de aplicații(asigură unei componente EJB o serie de funcționalități, ca injectarea dependențelor,conexiunea cu bazele de date, gestiunea tranzacțiilor)[1]

Tipuri de componente EJB:[7]

- *Session*:responsabile pentru un grup de funcționalități; de exemplu, o aplicație educațională ar putea avea un session bean pentru gestionarea notelor studenților și un alt bean pentru administrarea listei de cursuri și programe;
- *Message Driven*: preia mesaje de tipul Java Message Service (JMS).
- *Entity*: reprezintă datele ce vor fi persistate;

În continuare voi prezenta componenta EJB de tip session: Din punctul de vedere al reținerii stării, există următoarele tipuri de componente sesiune EJB:[35]

1. *stateless*-este tipul de ejb folosit pentru operații independente; după cum îi spune și numele, nu are asociată starea clientului, dar poate păstra starea inițială;containerul EJB construiește un *pool de instanțe* ale bean-urilor stateless pe care le folosește pentru a procesa cererile

clienţilor; de fiecare dată când un client accesează o metodă a bean-ului, o instanţa aleatoare este aleasă pentru a procesa acea cerere; acest lucru înseamnă că dacă un client face 2 cereri consecutive, este posibil ca 2 instanţe diferite să rezolve acea cerere; în cazul în care clientul dispare, instanţa nu este distrusă deoarece poate servi cererea următorului client;

**Exemplu:**

- (a) trimiterea unui e-mail ar putea fi manipulată de către un stateless bean, deoarece aceasta este o operaţiune ce nu face parte dintr-un proces cu mai mulţi paşi;
  - (b) un utilizator al unui site printr-un click pe un buton de tipul "ţine-mă la curent cu informaţii" poate declanşa un apel al unei metode asincrone al unui bean pentru a adăuga utilizatorul la o listă în baza de date a companiei;
  - (c) preluarea mai multor piese de date independente pentru un site web, cum ar fi o listă de produse sau istoria utilizatorului curent ar putea fi manipulate prin metode asincrone de un session bean;[35]
2. *stateful*-este tipul de bean care păstrează starea clientului, păstrează starea în variabilele instanţei; containerul EJB creează un bean separat pentru fiecare cerere a clientului; fiecare instanţă este ataşată unui singur client şi procesează doar cererile acelui client; deci dacă un client face 2 cereri consecutive, atunci cererea va fi procesată de aceeaşi instanţă; păstrează consistenţa datelor prin actualizarea câmpurilor obiectului la fiecare tranzacţie comisă;

**Exemplu:**

- (a) Controlul într-un magazin web ar putea fi manipulat de către un stateful bean, care ar folosi starea sa pentru a urmări starea clientului în procesul de control, eventual de a bloca elementele pe care clientul doreşte să le cumpere;[35]
3. *singleton*-există o singură instanţă a componentei EJB;durata de viaţă a componentei coincide cu intervalul de timp în care componenta EJB este activă în serverul de aplicaţii.

**Exemplu:**

- (a) încărcarea unei liste de preţuri zi cu zi, care va fi la fel pentru fiecare utilizator ar putea fi realizată cu un bean Singleton, deoarece acest lucru va împiedica aplicaţia să facă aceeaşi interogare pe o bază de date de foarte multe ori;[35]

## 7.2 JavaMail

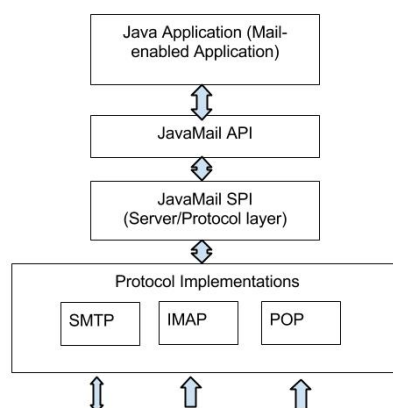
*JavaMail API* oferă un framework independent de platformă şi de protocol pentru a construi e-mailuri şi aplicaţii de mesagerie; API-ul oferă un set de clase abstracte care definesc obiecte care cuprind un sistem de e-mail; este un pachet opţional pentru citirea, compunerea şi trimiterea de mesaje electronice.

Protocoale suportate de API-ul JavaMail:

1. **SMTP- Simple Mail Transfer Protocol**, asigură un mecanism pentru trimitere de e-mail
2. **POP- Post Office Protocol**, mecanismul pe care mulţi îl folosesc pentru a primi e-mailuri;
3. **IMAP- Internet Message Access Protocol**, protocol avansat pentru primirea de mesaje; oferă suport pentru multiple căsuţe poştale pentru un singur utilizator;
4. **MIME- Multipurpose Internet Mail Extensions**, defineşte conţinutul a ceea ce este transferat, formatul mesajului, ataşamentele;
5. **NNTP-Network News Transfer Protocol**

### Arhitectura JavaMail:

Figura următoare ilustrează arhitectura API-ului:



Mecanismul abstract al API-ului este similar cu al altor API-uri J2EE, cum ar fi : JNDI, JDBC, şi JMS;

API-ul este împărţit în două mari părţi:

1. *o parte independentă de aplicaţie*: un API este folosit de componentele aplicaţiei pentru a trimite şi primi mesaje, indiferent de furnizor sau de protocolul utilizat;
2. *o parte dependentă de serviciu*: un SPI- *Service Provider Interface* înţelege limbajul specific protocolului utilizat; este folosit pentru a conecta furnizorul de servicii e-mail de platforma J2EE;

Pentru a trimite un e-mail se foloseşte serverul **SMTP: Simple Mail Transfer Protocol**; se pot folosi servere oferite de companii, cum ar fi yahoo sau gmail, sau se pot instala servere cum ar fi *Postfix*, pentru Ubuntu, *Apache James Server*, etc;[36]

## 7.3 IText

IText este o bibliotecă open-source ce permite crearea şi manipularea fişierelor PDF-*Portable Document Format*; oferă posibilitatea dezvoltatorilor ce doresc să sporească aplicaţii web, şi nu numai cu generare şi manipulare de fişiere PDF;

iText are o structură ierarhică; cea mai mică unitate de text este un *Chunk* care este un String cu un font predefinit. *Phrase* combină mai multe bucăţi şi permite definirea spaţierii între linii. *Paragraph* este o subclasă a clasei *Phrase*, şi permite definirea mai multor atribute de aspect, de exemplu, marginile. Clasa *Anchor*, este o subclasă a clasei *Paragraph* şi serveşte ca bază pentru hyperlink-uri în fişierul PDF generat.[37]

IText oferă următoarele facilităţi:

1. *generare documente şi rapoarte bazate pe date preluate din fişiere XML şi baze de date*
2. *adăugare pagini favorite, inscripţionări, numerotare de pagini*
3. *divizare şi concatenare de pagini din fişiere PDF existente*
4. *completare de formulare*
5. *oferă în mod dinamic fişiere PDF browserului web*
6. *fişiere PDF etichetate*
7. *adaugă semnături digitale unui fişier PDF*

## 7.4 JSON

JSON - *JavaScript Object Notation* este un format standard care foloseşte un limbaj uşor de înţeles de către oameni pentru a transmite obiecte constând din attribute cheie-valoare; de asemenea este uşor pentru calculatoare să parseze şi să genereze; Este folosit pentru a transmite date între un server şi o aplicaţie web, ca alternativă a limbajului XML.

Este un format de text **complet independent de limbaj**, dar foloseşte convenţii familiare programatorilor în limbaje cum ar fi: C, C++, C#, Java, JavaScript, Perl, Python şi multe altele;[38]

Este construit pe 2 structuri astfel;

- o colecţie de perechi nume-valoare; în multe limbaje poartă numele de *obiect*, *record*, *struct*, *dicţionar*
- o listă ordonată de valori; poate purta numele de *vector*, *listă*, *secvenţă*

### 7.4.1 GSON

*GSON* este o bibliotecă Java folosită pentru convertirea obiectelor Java în reprezentări JSON; de asemenea, converteşte string-uri JSON în obiecte Java; *GSON* poate lucra cu obiecte Java arbitrare, incluzând obiecte pre-existente pentru care nu există cod sursă;[39]

*Scopuri GSON*: [40]

1. oferă metodele *toJson()* şi *fromJson()* pentru convertirea obiectelor Java la JSON şi invers;
2. permite ca obiecte pre-existente să poată fi modificate în stringuri JSON;
3. suport pentru colecţii generice
4. permite reprezentări custom ale obiectelor
5. suportă obiecte complexe din punct de vedere arhitectural



## 7.5 JAXB

*JAXB -Java Architecture for XML Binding* oferă o modalitate rapidă şi convenientă de a lega scheme XML de reprezentări Java, facilitând munca dezvoltatorilor Java de a încorpora date XML în aplicaţii; JAXB oferă metode de citire a datelor XML în obiecte Java (*unmarshal*)şi de a scrie obiectele Java înapoi în fişiere XML(*marshal*);

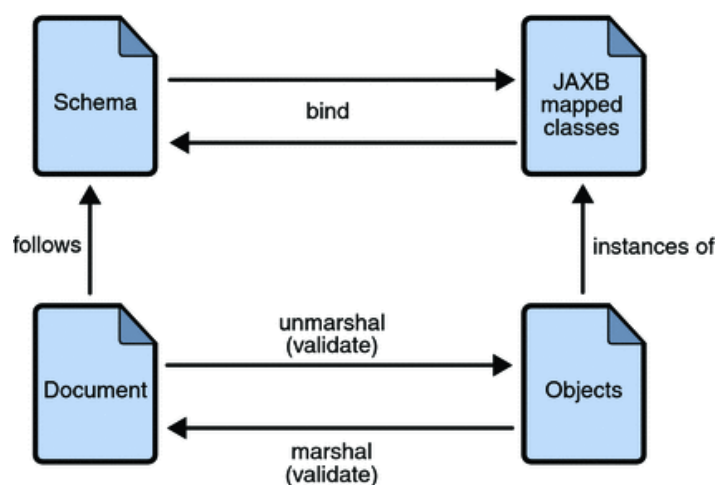


FIG. 7.1: procesul JAXB

### Procesul de legare în JAXB.Paşii principali

1. *generarea claselor*: o schemă XML este folosită ca input pentru compilatorul JAXB pentru a genera clase JAXB bazate pe acea schemă;
2. *compilarea claselor*: toate clasele generate, fişierele sursă şi codul aplicaţiei trebuie compilat;
3. *unmarshal*: documentele XML sunt citite de frameworkul JAXB;
4. *generarea arborelui de conţinut*: procesul de citire generează un arbore de obiecte instanţa a claselor generate;
5. *validarea-opţional*: procesul de citire implică validarea fişierului XML înainte de a genera arborele de conţinut;
6. *procesarea conţinutului*: aplicaţia client poate modifica datele din XML reprezentate ca obiecte Java;
7. *marshalling*: conţinutul procesat este scris în unul sau mai multe fişiere XML; conţinutul poate fi verificat înainte de scriere;

**Reprezentarea datelor XML în obiecte Java** Cel mai des folosite adnotări pentru citirea/scrierea datelor din/în fişiere XML:

Adnotare	Descriere
@XmlSchema	mapează un pachet la un XML namespace
@XmlAccessorType	controlează accesul la câmpurile şi proprietăţile clasei
@XmlAccessorOrder	controlează ordinea în care câmpurile vor fi scrise în fişierul XML
@XmlType	mapează o clasa Java la o anumită schemă
@XmlEnum	mapează o enumerare Java la o anumită schemă
@XmlRootElement	asociază un element global cu tipul schemei la care este mapată clasa
@XmlElement	mapează un câmp sau o proprietate la un element XML
@XmlElementWrapper	mapează o colecţie Java
@XmlAttribute	mapează o proprietate Java la un atribut XML
@XmlTransient	împiedică maparea proprietăţii la o reprezentare XML
@XmlID	mapează o proprietate Java la un XML id

## 7.6 Servicii REST

*REST -Representational State Transfer* este un stil arhitectural ce specifică anumite constrângeri care aplicate unui serviciu web induc proprietăţi dorite, cum ar fi **performanţă şi scalabilitate**, proprietăţi ce permit serviciilor să funcţioneze foarte bine.

În stilul REST, resursele şi funcţionalităţile sunt privite ca *resurse* şi sunt accesate folosind **Uniform Resource Identifiers -URI**, link-uri tipice; asupra resurselor acţionează operaţii simple şi bine definite; REST se rezumă la o arhitectură client-server ce foloseşte un **protocol stateless**, de obicei HTTPProtocolul HTTP; cu REST, clienţii şi serverul interschimbă reprezentări ale resurselor, folosind o interfaţăşi un protocol standard;[41]

Principiile stilului arhitectural REST:

1. **Identificarea resurselor prin URI:** un serviciu REST expune un set de resurse ce identifică obiectivele de interacţiune cu clientul;
2. **Interfaţă uniformă:** resursele sunt manipulate folosind un set de 4 operaţii: *PUT, GET, POST, DELETE*;
3. **Mesaje auto-descriptive:** resursele sunt decuplate de reprezentările lor, astfel conţinutul lor poate fi accesat într-o varietate de formate, cum ar fi JSON, HTML, XML, etc;

4. **Interacţiuni stateful prin intermediul link-urilor:** fiecare interacţiune cu o resursă este stateless; există anumite mijloace de a reţine starea, cum ar fi rescrierea URL-urilor, cookie-urile sau câmpuri ascunse într-un formular;[41]

### 7.6.1 Dezvoltarea serviciilor REST folosind JAX-RS

JAX-RS este un API Java folosit pentru dezvoltarea aplicaţiilor folosind stilul arhitectural REST; API-ul foloseşte adnotările limbajului de programare Java pentru simplifica procesul de dezvoltare al serviciilor REST; programatorii decorează clasele Java cu adnotări JAX-RS pentru a defini resursele şi operaţiunile ce se pot efectua asupra lor; adnotaţiile JAX-RS sunt adnotaţii *runtime*; astfel prin *reflecţie* se vor genera clase ajutătoare şi artefacte pentru resursă; [8]

Adnotări JAX-RS des folosite:

Adnotare	Descriere
@Path	indică URI-ul unde clasa Java va fi găzduită
@GET	corespunde metodei similare HTTP; o metodă Java adnotată astfel va procesa toate cererile HTTP GET;
@POST	corespunde metodei similare HTTP; o metodă Java adnotată astfel va procesa toate cererile HTTP POST;
@DELETE	corespunde metodei similare HTTP; o metodă Java adnotată astfel va procesa toate cererile HTTP DELETE;
@PUT	corespunde metodei similare HTTP; o metodă Java adnotată astfel va procesa toate cererile HTTP PUT;
@HEAD	corespunde metodei similare HTTP; o metodă Java adnotată astfel va procesa toate cererile HTTP HEAD;
@PathParam	se foloseşte pentru injectarea valorilor din URL în parametri unei metode; parametri de cale sunt extraşi din URI şi numele parametrilor corespund cu numele variabilelor specificate în adnotarea @Path
@QueryParam	parametri de interogare sunt extraşi din URI-ul cererii;
@Consumes	folosită pentru a specifica tipul MIME al unei reprezentări pe care o resursă îl poate accepta şi consuma
@Produces	folosită pentru a specifica tipul MIME al unei reprezentări pe care o resursă îl poate produce şi trimite înapoi clientului
@ApplicationPath	este folosită pentru a defini URL ce mapează aplicaţia; calea specificată de adnotare este URI-ul de bază pentru toate URI-urile resurselor specificate în adnotarea @Path;

## 7.7 Comet

**Comet** este un model de aplicaţie web în care o cerere HTTP permite unui web server să împingă informaţie într-un browser fără ca acesta să ceară în mod explicit; Comet este un *termen umbrelă* cuprinzând multe tehnici pentru a realiza această interacţiune; toate aceste metode se bazează pe caracteristici incluse în browsere, cum ar fi *JavaScript*, mai mult decât pe plugin-uri; această abordare diferă de modelul web original, în care browserul cere o pagină la un moment dat.

Comet este cunoscut şi sub alte nume, incluzând printre altele *Ajax Push*, *Reverse Ajax*, *Two-way-web*, *Reverse Steaming*, *HTTP Server Push*;

### Motivarea folosirii Comet

Protocolul HTTP reprezintă fundaţia schimbului de informaţii pe Internet; totuşi are câteva limitări, de exemplu este un protocol *stateless*, *one-way*; o cerere este trimisă unui server, iar serverul trimite înapoi răspunsul; cererea trebuie iniţiată de client, şi numai serverul poate răspunde cererii; acest aspect face ca un număr de aplicaţii web să fie cel puţin impractice; un exemplu elocvent ar fi un *chat*, aplicaţie folosită chiar în proiectul de faţă;

Comet oferă o bună îndepărtare faţa de modelul de comunicare HTTP, permiţând un stil *push*; Comet defineşte mai multe tehnici care permit serverului să trimită informaţii browserului fără intervenţia clientului; cu ajutorul unei conexiuni HTTP adiţionale, Comet poate facilita comunicare bi-direcţională; [43]

### Implementări Comet. CometD,

Există câteva framework-uri ce simplifică dezvoltarea de aplicaţii pe baza modelului Comet; cel mai notabil este *CometD*, ce implementează specificaţiile protocolului *Bayeux*;

Bayeux este un protocol folosit pentru transmiterea mesajelor asincrone cu latenţă redusă între un web server şi un client; mesajele pot fi livrate:

- de la server la client
- de la client la server
- de la client la alt client

Bayeux caută să reducă complexitatea dezvoltării unei aplicaţii web bazate pe modelul Comet, permiţând programatorilor să rezolve uşor problemele legate de distribuirea mesajelor;[44]

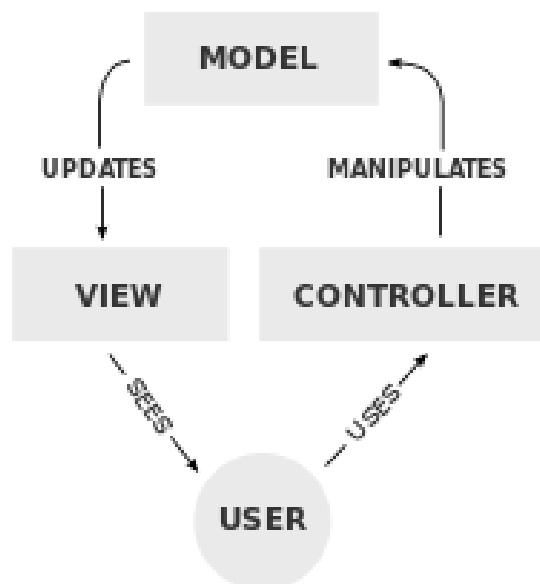
CometD este un proiect open-source dezvoltat de fundaţia *Dojo* ce permite transportarea mesajelor în aplicaţii web bazate pe Ajax între un client şi un server, şi mai important între server şi client;

## Capitol 8

### Structura aplicației.MVC

Aplicația este structurată în 3 mari proiecte: model, view, service pe baza modelului dat de design-pattern-ul *MVC Model-View-Controller*.

Model-View-Controller este un șablon clasic folosit în aplicații care necesită o separare clară între partea de *business logic* și *interfață*; MVC izolează logica aplicației de interfață, permițând dezvoltare individuală, testare și întreținere pentru fiecare componentă în parte.



Aici Controller-ul primește toate cererile de la aplicație și apoi lucrează cu modelul pentru a pregăti toate datele necesare View-ului; apoi View-ul folosește datele pregătite de Controller pentru a genera un răspuns final.

Ideea este de a face o distincţie clară între obiecte ce modelează percepţia noastră a lumii reale, şi obiecte de prezentare, care sunt elementele GUI pe care le vedem pe ecran; obiectele ar trebui să fie complet autonome şi să lucreze fără a face referire la prezentare, ele ar trebui de asemenea, să ofere posibilitatea de a sprijini mai multe prezentări, eventual în acelaşi timp.

Acest şablon este divizat în trei părţi:

1. *Model*: această componentă administrează informaţiile şi notifică observarii când informaţia se schimbă; reprezintă datele asupra cărora aplicaţia operează; asigură stocarea persistentă a datelor care sunt manipulate de către *controller*; răspunde cererilor din partea view-ului şi răspunde, de asemenea instrucţiunilor din partea controller-ului;
2. *View*: componenta afişează datele şi de asemenea preia date de la utilizator; interpretează datele din model într-o formă pe care o transmite utilizatorului; pot fi mai multe componente view asociate unui singur model; de fapt, este o reprezentare a datelor din model;
3. *Controller*: manipulează toate cererile primite de la view sau de la interfaţa utilizatorului; întreg flux de date al aplicaţiei este controlat de controller; doar controller-ul poate accesa datele din model şi doar el poate trimite date spre variaţi clienţi; este responsabil pentru a răspunde la datele introduse de utilizator şi de a efectua operaţii pe obiectele din model; primeşte datele de intrare, validează inputul iar apoi efectuează operaţiile care modifică starea modelului.[20]

Deşi iniţial dezvoltat pentru aplicaţii desktop, şablonul a fost adoptat pe scară largă de limbajele de programare majoră ca o arhitectură pentru aplicaţii web.[21]

## Capitol 9

# Aplicația rezultată

### 9.1 Prezentarea bazei de date Shop4j

Cum am menționat și în secțiunea MySQL, pentru a persista datele aplicației am ales crearea unei baze de date în sistemul de gestiune *MySQL*;

Conexiunea la baza de date se face prin intermediul fișierului de configurare *standalone.xml*, care se găsește în următoarea ierarhie de directoare: *jboss-as-7.1.1.Final/standalone/configuration/*; conexiunea la baza de date prin intermediul fișierului de configurare reprezintă încă un avantaj al unui server de aplicație față de un web server;

Codul folosit pentru conexiune este următorul:

```
<connection-url>
.....jdbc:mysql://localhost:3306/shop4j
</connection-url>
<driver>mysql</driver>
<security>
.....<user-name>root</user-name>
.....<password>*****</password>
</security>
```

După cum se vede din atributul *connection-url*, baza de date, numită *shop4j* se găsește la hostul *localhost*, pe portul *3306*;

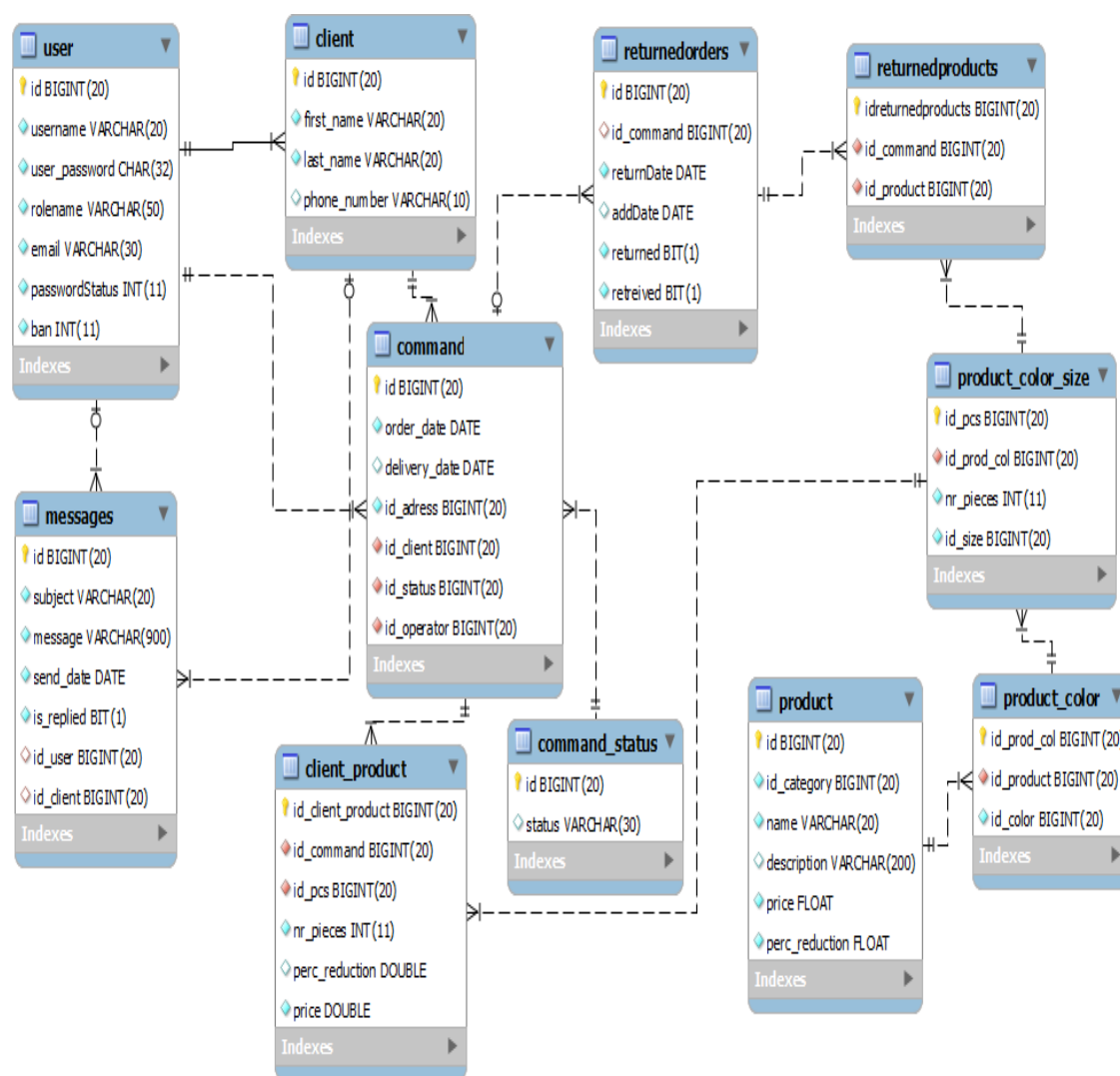
Username-ul folosit pentru conexiune este cel implicit, *root*, dar este setată o parolă pentru sporirea securității;

Baza de date este compusă din 9 tabele:

Nume tabel	Descriere
<i>user</i>	păstrează username-ul şi parola contului, mail-ul utilizatorului, precum şi rolul acestuia: <i>client sau operator</i> ; de asemenea, păstrează şi statusul parolei folosit în cazul în care utilizatorul îşi uită parola; pe baza datelor din acest tabel se realizează autentificarea utilizatorilor;
<i>client</i>	reţine date despre client: nume, prenume şi număr de telefon;
<i>command</i>	reţine date despre comandă: data comenzii, data livrării (în cazul în care a fost livrată), statusul comenzii, clientul care a iniţiat comanda, adresa la care va fi livrată comanda şi operatorul care se va ocupa de finalizarea ei;
<i>commandstatus</i>	statusul unei comenzi poate fi : <i>in progress, delivered</i> ; aceste date se folosesc pentru informarea clientului în legătură cu comanda
<i>clientproduct</i>	pentru fiecare produs al unei comenzi se reţin numărul de bucăţi dorite, procentul de reducere, pentru a se calcula preţul total al comenzii;
<i>product</i>	în acest tabel se reţin date despre fiecare produs din stoc: nume, descriere, categoria din care face parte, şi evident preţul;
<i>returnedorders</i>	tabelul foloseşte salvării comenzilor pe care clienţii doresc să le returneze dintr-un motiv sau altul ;
<i>returnedproducts</i>	pentru fiecare comandă returnată, se reţin doar produsele pe care clientul doreşte să le returneze
<i>messages</i>	aici se păstrează toate mesajele şi comentariile pe care clienţii le lasă referitor la magazinul online; doar clienţii pot lăsa mesaje, iar în cazul în care operatorul consideră că mesajul este nepotrivit, clientului îi va fi interzis accesul la aplicaţie;



În figura următoare este prezentată o diagramă EER - *Enhanced Entity Relationship* creată în aplicaţia *MySQL Workbench* ce conţine tabelele bazei de date:



Cum am spus şi în secţiunea JPA, toate tabelele sunt mapate la clase Java; acest lucru este realizat de Hibernate prin intermediul adnotărilor; Figura următoare reprezintă o diagramă UML - *Unified Modeling Language* a claselor rezultate din maparea tabelelor;

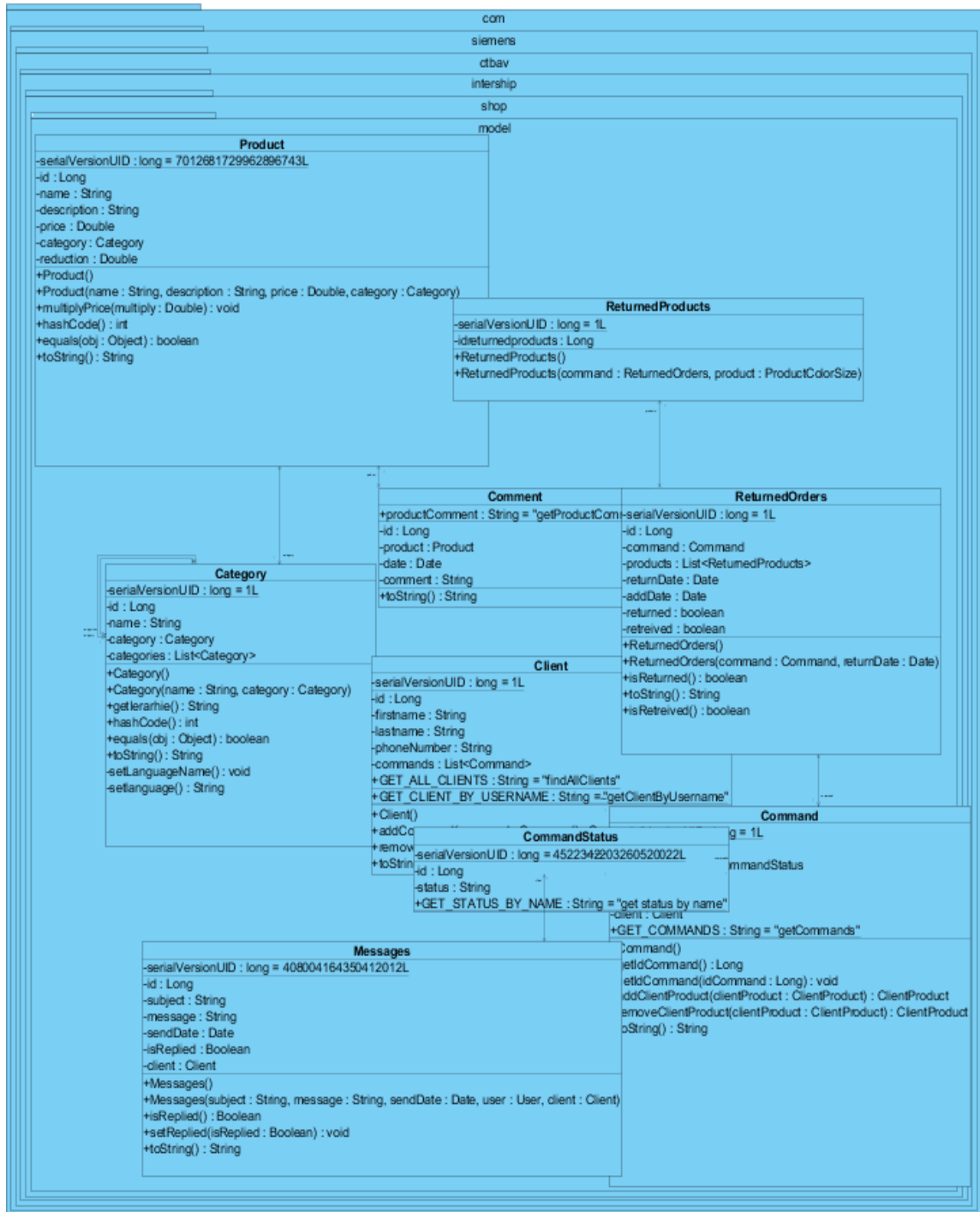


FIG. 9.1: Diagrama UML reprezentând clasele model

Pentru această aplicaţie am ales să nu folosesc direct entităţile generate cu Hibernate; motivul este că de multe ori în stratul de prezentare aveam nevoie ca datele să fie formate într-un anumit fel, diferit de formatul entităţilor.

Pentru aceasta am folosit clase **DTO -Data Transfer Object**: clasele DTO sunt clase ce expun proprietăţi, fără metode folosite pentru a asigura izolare între diferite părţi ale aplicaţiei; sunt folosite când se doreşte gruparea datelor în structuri pentru transmiterea lor; DTO-urile ajută decuplarea stratului de prezentare de partea de service şi de clasele model;[45]

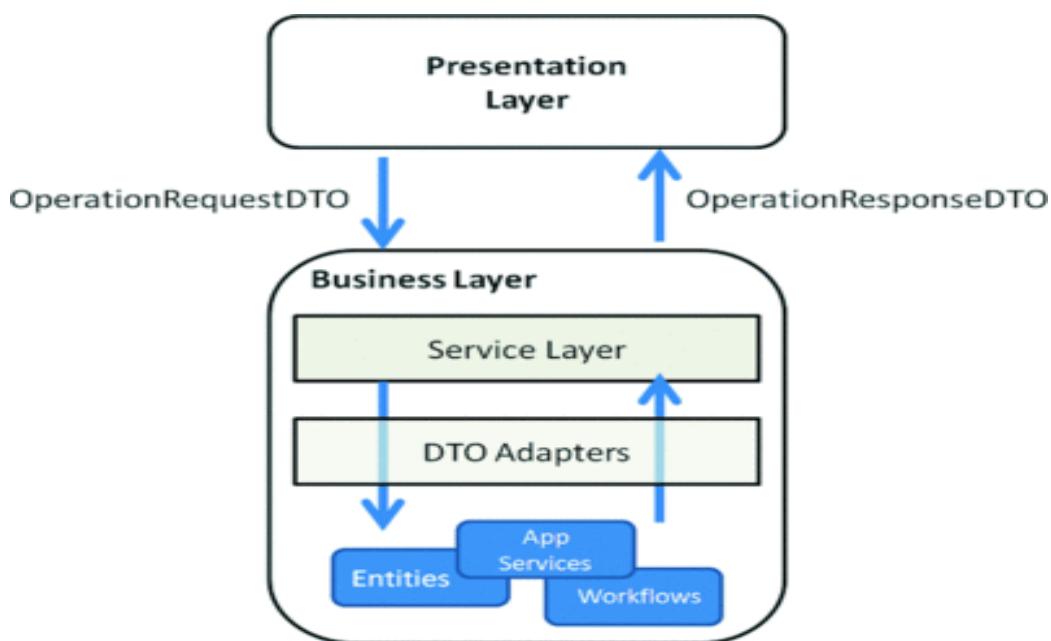


FIG. 9.2: DTO

Pentru transformarea unui obiect entitate într-un obiect DTO am folosit așa numitele *clase convertor*; de exemplu, pentru clasa *ReturnedProducts*, se aplează metoda statică de conversie, *convertReturnedProduct* din clasa *ConvertReturnedProduct*

```
public static ReturnedProductsDTO
    convertReturnedProduct(ReturnedProducts prod){
if(prod == null) return new ReturnedProductsDTO();
ProductColorSizeDTO product=ConvertProductColorSize.
    convert(prod.getProduct());
ReturnedOrdersDTO comm = new ReturnedOrdersDTO();
return new ReturnedProductsDTO(prod.
    getIdreturnedproducts(),comm, product);
}
```

## 9.2 Funcţionalităţi

În această aplicaţie actorul principal, anume, *operatorul* se ocupă cu gestiunea clienţilor şi a comenzilor; în figura următoare, ce reprezintă o diagramă UML se pot vedea grafic funcţionalităţile operatorului:

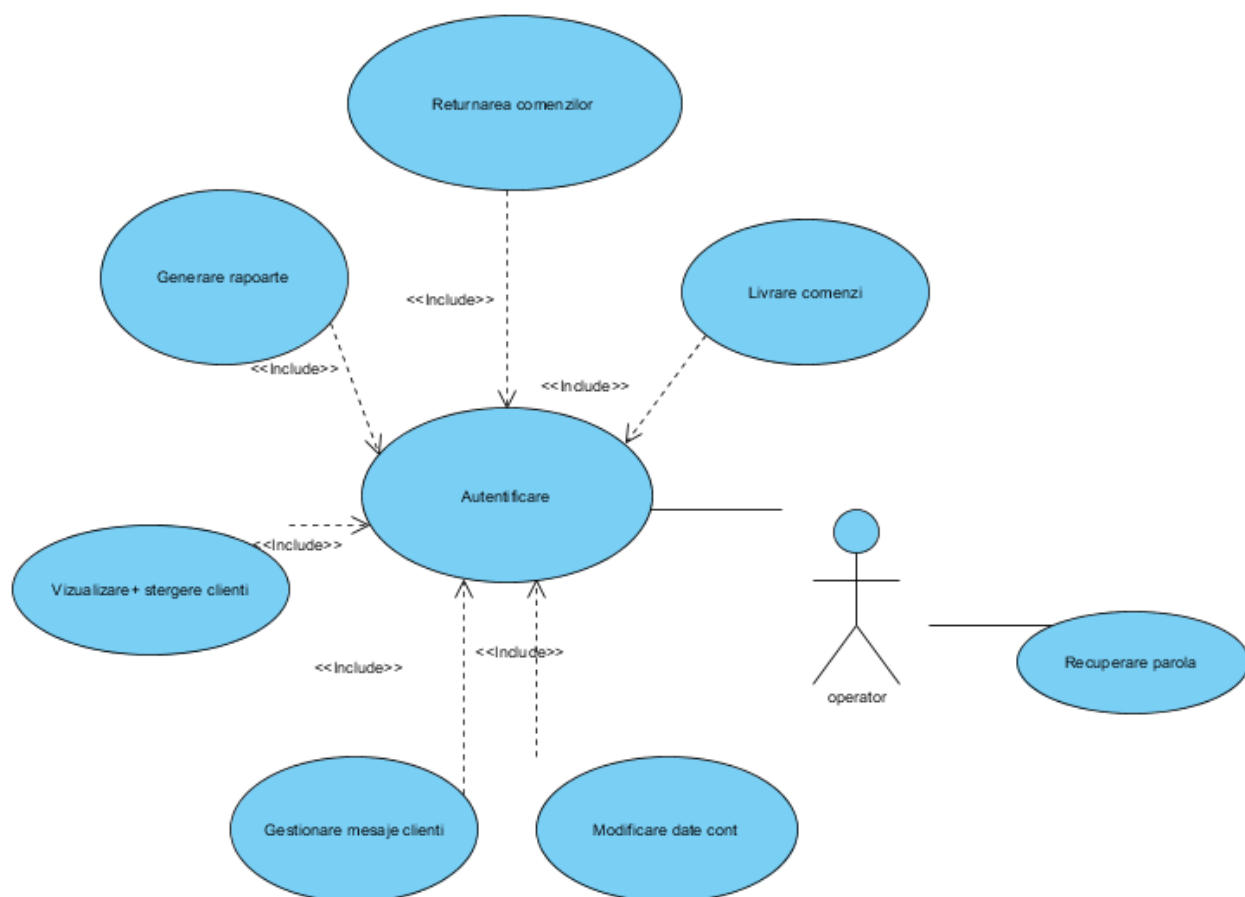


FIG. 9.3: UML-Use case

### 9.2.1 Schimbare cont

Operatorul are mereu posibilitatea de a-şi actualiza datele personale din baza de date; de exemplu, dacă are o nouă adresă de e-mail sau dacă doreşte să îşi modifice username-ul sau/şi parola din motive de securitate, doar accesează linkul *Change Account*; această pagină conţine un formular pe care operatorul îl completează iar apoi apasă butonul pentru a salva datele.

Dacă doreşte să modifice parola, o clasă ce implementează interfaţa *javax.faces.validator.Validator* va verifica dacă parola nou introdusă este validă; în cazul în care cerinţele nu sunt respectate, operatorul va primi un mesaj de eroare; mesajul primit explică foarte clar ceea ce s-a greşit astfel încât operatorul să poată completa cu date valide;

Codul următor este preluat din pagina *changeAccount.xhtml*, reprezintă secţiunea de verificare a validităţii parolei;

FIG. 9.4: Change account error

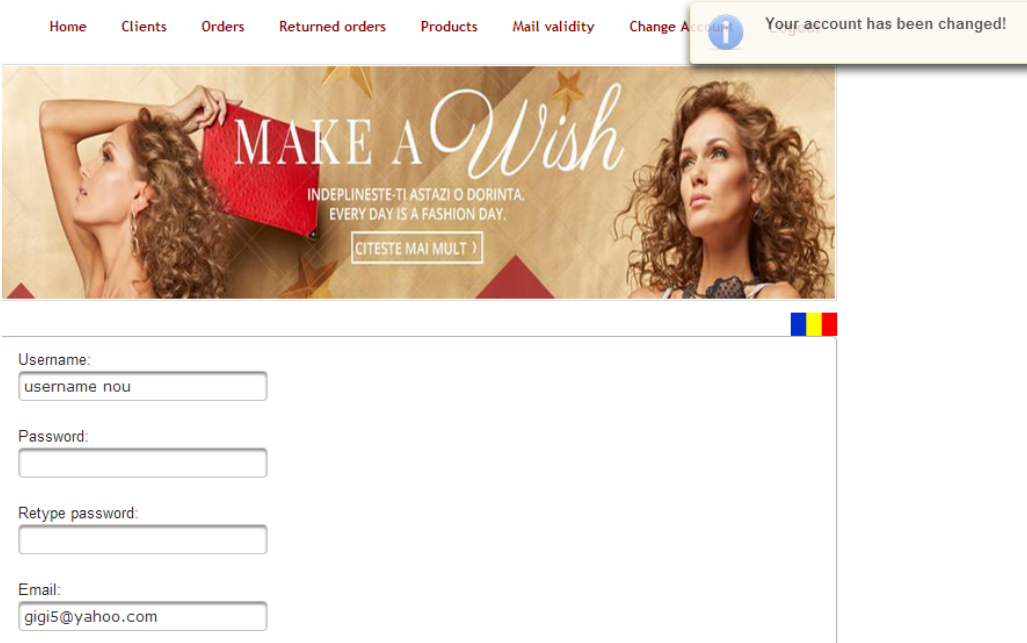
```
<p:password id="password" value="#{
    changeAccount.user.password}"
    label="Password" required="true">
<f:validator validatorId="validatePassword" />
    <f:attribute name="confirm" value="#{confirm}" /
    >
</p:password>
<p:message for="password" />
    <h:outputLabel for="confirm" value="Retype
        password: " />
    <p:password id="confirm" value="#{
        changeAccount.user.retypePassword}"
        label="Retyped password" binding="#{confirm}"
        required="true" />
<p:message for="confirm" />
```

După ce operatorul introduce noile date în formular, se verifică dacă prin inserarea lor se produc duplicate în baza de date; evident, nu este permis să existe 2 valori identice ale username-ului (pentru că s-ar genera probleme la logare) sau ale e-mailului.

Secvenţa următoare de cod prezintă exact verificarea duplicităţii username-ului şi a adresei de e-mail:

```
Long id = getUserId(oldUsername);  
if (!usernameAlreadyExists(oper.getUsername(), id) && !  
    emailAlreadyExists(oper.getEmail(), id)) {  
    User user = em.find(User.class, id);  
    user.setUsername(oper.getUsername());  
    user.setEmail(oper.getEmail());  
    user.setUserPassword(oper.getPassword());  
    em.merge(user);  
}
```


Dacă valorile preluate din formular respectă constrângerile, atunci se efectuează modificările în bază de date şi operatorul este înştiinţat că modificările s-au efectuat cu succes;



The screenshot displays a web application interface. At the top, a navigation bar contains links: Home, Clients, Orders, Returned orders, Products, Mail validity, and Change Account. A notification box on the right states "Your account has been changed!". Below the navigation bar is a banner for "MAKE A Wish" with the text "INDEPLINEŞTE-ŢI ASTĂZI O DORINŢĂ. EVERY DAY IS A FASHION DAY." and a button "CITEŞTE MAI MULT". The main content area features a registration form with the following fields: Username (containing "username nou"), Password, Retype password, and Email (containing "gigi5@yahoo.com"). A small Romanian flag is visible on the right side of the form.

## 9.2.2 Vizualizare clienţi

Operatorul deţine lista tuturor clienţilor activi din baza de date; acesta poate vizualiza datele clienţilor: numele, prenumele, numărul de telefon şi data ultimei comenzi; de asemenea, poate accesa şi date mai importante, cum ar fi email-ul sau username-ul pe care îl foloseşte când se loghează;



Expand rows to see detailed information									
	First name	Last name	Phone number	Last order date					
▼	Gigi	Muresan	0712345678	2014-04-04 00:00:00.0	Delete				
<table><tr><td>Username:</td><td>client1</td></tr><tr><td>Email:</td><td>gigi@yahoo.com</td></tr></table>						Username:	client1	Email:	gigi@yahoo.com
Username:	client1								
Email:	gigi@yahoo.com								
▶	Moise	Robert	0789453212		Delete				

FIG. 9.5: Vizualizare + ştergere clienţi

În cazul în care clientul nu a mai comandat de foarte mult timp sau nu a comandat deloc, operatorul îi poate şterge contul; acest lucru se întâmplă pentru a nu suprapopula baza cu date nefolositoare; evident, dacă pe viitor doreşte, clientul îşi poate crea un nou cont;

```
public void deleteUser(Long id) throws UserException {  
    User u = em.find(User.class, id);  
    if (u == null) throw new UserException(  
        internationalizationService.getMessage("userNotFound"  
        ") + id);  
    em.remove(u);  
}
```

În cazul în care există comenzi iniţiate de clientul ce se doreşte a fi şters, operatorul primeşte un mesaj de eroare ce îl anunţă că nu are permisiune de ştergere;

Clientul va fi notificat de ştergerea contului printr-un mail;

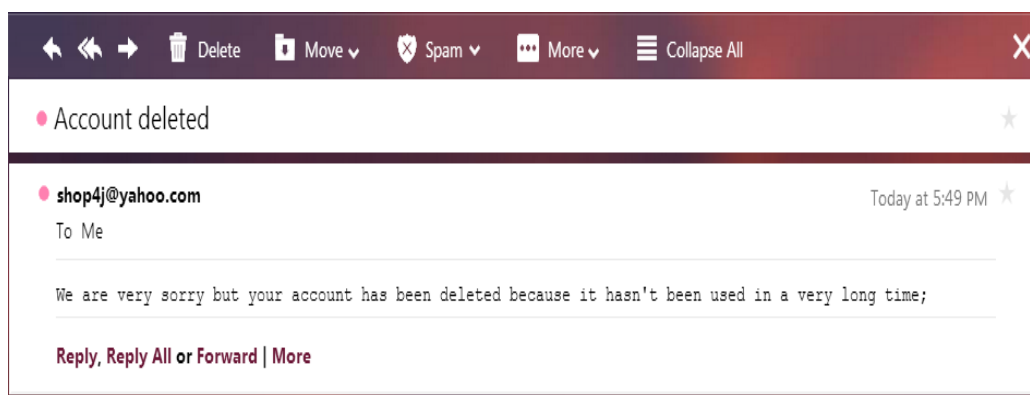


FIG. 9.6: Notificarea clientului prin mail

### 9.2.3 Generare rapoarte

Operatorul poate genera 4 tipuri diferite de rapoarte: *PDF*, *JSON*, *XML* şi *CSV* (*Comma separated values*).

Aceste rapoarte pot fi generate din pagina clienţilor sau din pagina de comenzi; operatorul poate alege mai multe tipuri de rapoarte, le poate alege chiar pe toate; rapoartele se vor salva pe calea *D:/Reports*, fiecare cu un nume sugestiv, de exemplu: *Client09.06.2014*;

☐ PDF ☐ XML ☐ JSON ☐ CSV

**Generate my reports, please**

FIG. 9.7: Selectare tipuri de rapoarte

Pentru generarea fiecărui tip de raport am folosit o tehnologie diferită: pentru rapoarte de tip *PDF* (*Portable Document Format*) am folosit biblioteca *IText*, pentru fişierele *CSV* am folosit doar clase Java deja existente, pentru fişiere *XML* am folosit *JAXB* şi nu în ultimul rând pentru rapoartele *JSON*, am folosit biblioteca *GSON*.



Un raport JSON arată astfel:

```
1 {
2   "firstName": "Gigi", "lastName": "Muresan", "phoneNumber
   ": "0712345678",
3     "user":
4       { "username": "client1", "email": "gigi@yahoo.
         com",
5         "password": "client1", "rolename": "client",
6         "passwordStatus": "SAVED"
7       },
8     "lastOrderDate": "Jan 1, 2014 12:00:00 AM"
9 }
```

Pentru generarea rapoartelor XML am folosit JAXB; acelaşi raport ca cel de mai sus, dar în format XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"
?>
- <list xsi:type="clientJAXB" xmlns:xsi="http://www.w3.
  org/2001/XMLSchema-instance">
  <<firstName>Gigi</firstName>
  <<lastName>Muresan</lastName>
  <<lastOrderDate>2014-01-01T00:00:00+02:00</
    lastOrderDate>
  <<phoneNumber>0712345678</phoneNumber>
  -<user>
  <<email>gigi@yahoo.com</email>
  <<password>client1</password>
  <<passwordStatus>SAVED</passwordStatus>
  <<rolename>client</rolename>
  <<username>client1</username>
  <</user>
  <</list>
```

## 9.2.4 Actualizare stoc produse

În cazul în care nu mai sunt destule produse în stoc pentru a onora toate comenzile, operatorul are responsabilitatea de a actualiza stocul; când se primesc produse, operatorul adaugă exact numărul de produse primite în baza de date, așa cum se poate vedea și în figura următoare;

Home Clients Orders Returned orders Products Mail validity Change Account Logout

Name	Description	Color	Size	Category	Missing pieces	Pieces I want to add	
V TShirt	cotton 100%	Red	xs	TShirt	11	<input type="text" value="5"/>	<b>Add products</b>

(1 of 1) << < 1 > >>

FIG. 9.8: Actualizare stoc produse

În secvența de cod de mai jos metoda *addProducts* ce primește ca parametru produsul pentru care se dorește incrementarea stocului caută în baza de date produsul din parametru, și adaugă numărul de bucăți din parametru la numărul de produse deja existente în stoc; metoda aruncă o excepție de tipul *ProductException* în cazul în care produsul nu este găsit în baza de date:

```
@SuppressWarnings("unchecked")
public void addProducts(MissingProduct pr)
    throws ProductException {
    ProductColorSizeDTO product = pr.getProduct();
    List<ProductColorSize> prod = em
        .createNamedQuery(ProductColorSize.
            GET_PRODUCT_COLOR_SIZE)
        .setParameter("color",
            product.getProdColor().getColor().
                getName())
```

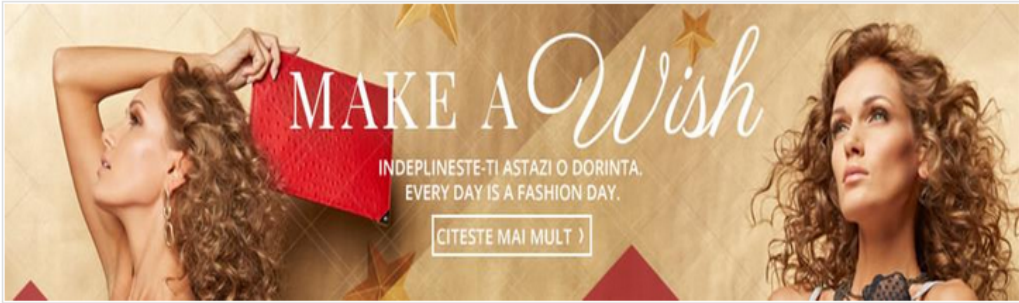
```


        .setParameter("size", product.getSize().getName()
        ())
        .setParameter("name",
            product.getProdColor().getProduct().
            getName())
        .getResultList();
prod.get(0).setNrOfPieces(
    prod.get(0).getNrOfPieces() + pr.getPiecesAdded
    ());
em.merge(prod.get(0));
}

```

După ce se modifică datele în bază, pagina se reîncarcă şi operatorul poate vedea de câte produse ar mai fi nevoie pe viitor; în cazul în care numărul de produse din stoc este mai mare decât numărul de produse necesare, în tabel, numărul va fi negativ;

Cum se vede şi în figura de mai jos, mai sunt nevoie de 6 bucăţi din produsul *V TShirt* pe culoarea roşu, dar sunt 4 bucăţi în plus din produsul *V TShirt* pe culoarea verde:





Name	Description	Color	Size	Categ	Missing pieces	Pieces I want to add	
V TShirt	cotton 100%	Red	xs	TShirt	6	<input type="text"/>	<b>Add products</b>
V TShirt	cotton 100%	Green	s	TShirt	-4	<input type="text"/>	<b>Add products</b>

(1 of 1)

FIG. 9.9: Actualizare stoc produse

## 9.2.5 Recuperare parola

Există cazuri când operatorul uită parola de la propriul cont; în aceste ocazii, foloseşte adresa de e-mail pentru a-şi recupera contul; operatorul îşi introduce e-mailul în formular, iar apoi apasă butonul *Ok*;




FIG. 9.10: Forgot password

Aplicaţia caută în baza de date pentru a valida e-mailul; dacă nu găseşte va afişa un mesaj de eroare; în cazurile când validarea are un rezultat pozitiv, se generează o parolă random din 10 caractere, unică care se criptează cu cifrul *AES*; tot aici, se criptează şi data curentă, cu acelaşi cifru(data este folosită pentru verificarea validităţii link-ului); aceste informaţii înglobează într-un URL şi se trimit operatorului pe e-mail;

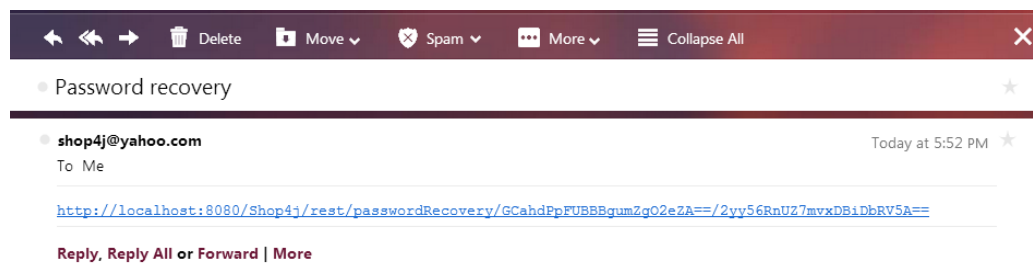


FIG. 9.11: Link

Parola nou generată va fi salvată în baza de date şi va avea statusul *NEW-GENERATED*; contul operatorului va fi blocat până când va accesa link-ul trimis pe mail;

id	username	user_password	rolename	email	passwordStatus	ban
1	delia	JG17QN36YA	client	deliutzzz_23@yahoo.com	0	0

FIG. 9.12: Changes in database

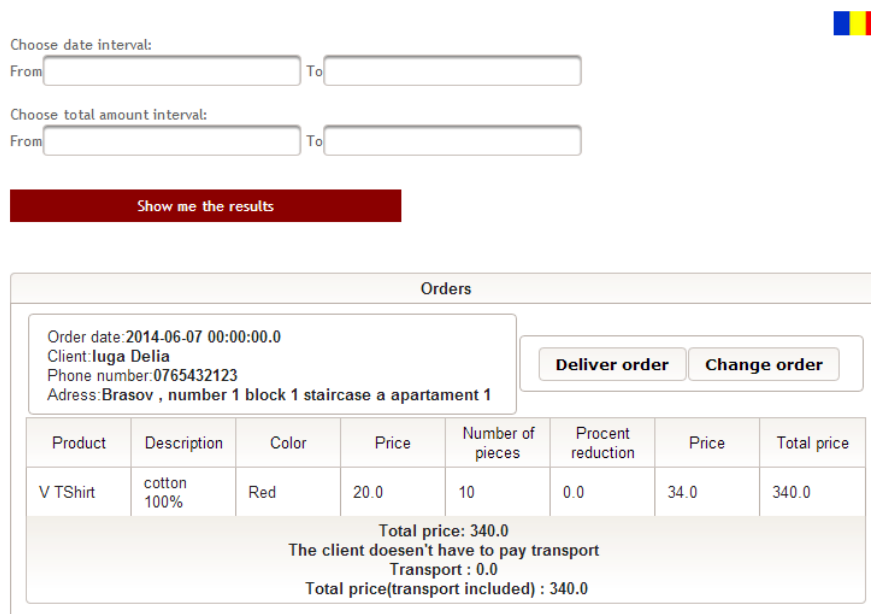
Operatorul va accesa acest link şi în cazul în care nu a trecut o perioadă mai mare de 15 zile, va fi trimis pe o pagină unde îşi va putea seta o nouă parolă; preluarea parolei şi a datei din URL se face cu ajutorul unui serviciu *REST*:

```
@GET
@Path("/{ milis }/{ password }")
public Response setNewPassword(@PathParam(" milis ")
    String milis ,
    @PathParam(" password ") String password ,
    @Context HttpServletRequest request ,
    @Context HttpServletResponse response) {
    String contextPath = request.getContextPath();
    String decryptedPass = null;
    Long time = null;
    String decryptedTime = null;
    try {
        decryptedPass = AES.decrypt(milis);
        decryptedTime = AES.decrypt(password);
        time = Long.parseLong(decryptedTime);
    } catch (Exception exc) {
        return Response.status(400).entity(password).
            build();
    }
    if (isValidLink(time) == false) {
        redirectTo(response , contextPath , "/login.xhtml
            ");
        return Response.status(400).entity(milis).build
            ();
    }
    redirectTo(response , contextPath , "/changePassword.
        xhtml?password="
            + milis);
    return Response.status(Status.ACCEPTED).build();
}
```

Acum operatorul va putea seta o nouă parolă; dacă parola va trece de verificări, va fi salvată în baza de date şi operatorul va putea accesa aplicaţia din nou;

### 9.2.6 Livrare comenzi

Operatorul se ocupă de gestiunea şi livrarea comenzilor; fiecare operator are un număr de maxim 10 comenzi de care se ocupă la un moment dat; în momentul în care comanda este livrată operatorul primeşte o nouă comandă; de asemenea, operatorul poate sa renunţe la o comandă în cazul în care nu doreşte să se ocupe de ea; evident, va primi alta la schimb;



Choose date interval:  
From  To

Choose total amount interval:  
From  To

**Show me the results**

**Orders**

Order date: 2014-06-07 00:00:00.0  
Client: Iuga Delia  
Phone number: 0765432123  
Adress: Brasov , number 1 block 1 staircase a apartament 1

**Deliver order** **Change order**

Product	Description	Color	Price	Number of pieces	Procent reduction	Price	Total price
V TShirt	cotton 100%	Red	20.0	10	0.0	34.0	340.0

Total price: 340.0  
The client doesn't have to pay transport  
Transport : 0.0  
Total price(transport included) : 340.0

FIG. 9.13: Orders list

După cum se vede şi în figura de mai sus, operatorul poate filtra comenzile în funcţie de costul total sau de data în care a fost iniţiată;

În momentul în care doreşte să livreze o comandă şi apasă pe butonul *Deliver*, se verifică în baza de date dacă există destule produse pe stoc pentru a fi onorată comanda; în cazul nefericit în care sunt prea puţine produse, un mail este trimis clientului în care îi este explicată situaţia;

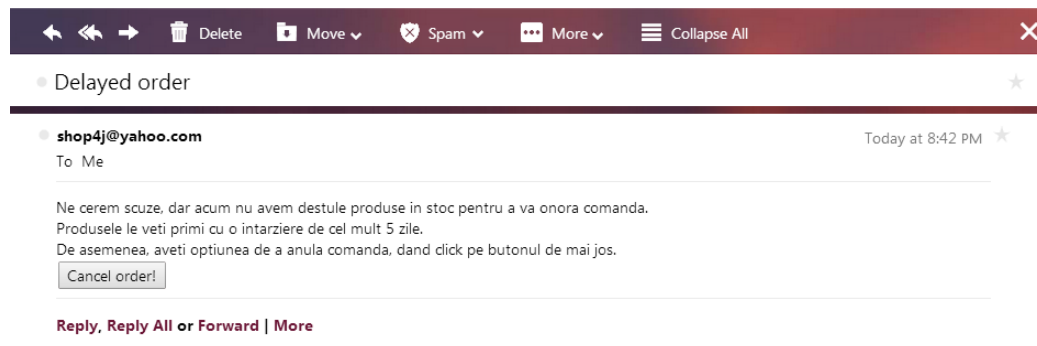


FIG. 9.14: Cancel order

Dacă există destule produse, deci comanda poate fi onorată, un mail se trimite clientului, pentru a-l înştiinţa că produsele cerute sunt pe drum;

În cazul în care clientul decide că nu doreşte să aştepte mai multe zile, apasă butonul *Cancel order* şi este trimis pe o pagină unde poate anula comanda; acest lucru se face tot cu ajutorul unui serviciu *REST*. Dacă clientul nu s-a răzgândit şi mai doreşte sănuleze comanda, apasă butonul *Yes* din formular şi comanda va fi ştearsă din baza de date;

Cum am mai spus, operatorul poate schimba o comandă în cazul în care nu vrea să se ocupe de ea, apăsând butonul *Change order*; prima dată, se caută în bază o comandă neasignată niciunui operator, şi se asignează operatorului care doreşte schimbul; dacă nu exista nicio astfel de comandă, operatorul primeşte un mesaj de eroare; al doilea pas este eliberarea comenzii anterioare:

```
@SuppressWarnings("unchecked")
public void assignOperatorsOrder(long id_operator)
    throws CommandNotFoundException,
        UserNotFoundException {
    List<Command> comm = em.createNamedQuery(Command.
        GET_ORDER).setMaxResults(1).getResultList();
    if (comm == null || comm.size() == 0) throw new
        CommandNotFoundException("No_more_unassigned_orders");
    User user = em.find(User.class, id_operator);
    if (user == null) throw new UserNotFoundException("No_
        operator_with_this_id:" + id_operator);
    comm.get(0).setUser(user);
}
```

### 9.2.7 Comenzi returnate

În cazul în care un client se simte nemulţumit de un anumit număr de produse dintr-o comandă le poate returna; pentru aceasta, intră pe cont la secţiunea *Command history*, selectează comanda iar apoi produsele pe care doreşte săle returneze; o comandă se poate returna doar dacă nu au trecut mai mult de 20 de zile de la primirea ei;

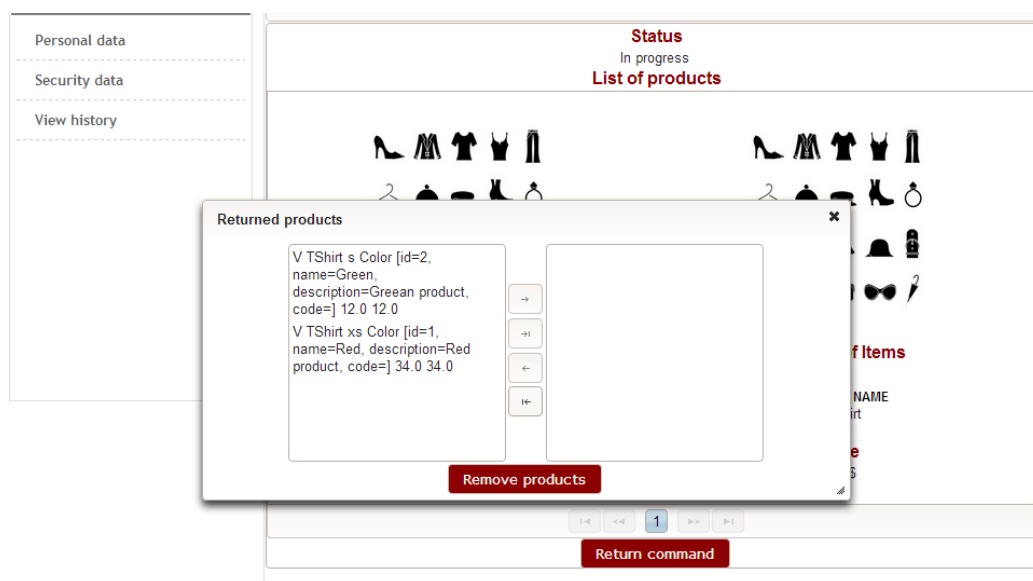


FIG. 9.15: Return order

După ce selectează comanda, clientul are de ales din *pick list* produsele pe care doreşte să le returneze; apoi, apasă butonul *Remove products* şi cererea este înregistrată; după salvarea cererii în bază clientul este înştiinţat printr-un mesaj;

În figura de mai jos se observă înregistrarea corespunzătoare comenzii ce se doreşte a fi returnată; similar arată şi înregistrarea corespunzătoare produselor returnate;

Filter:				Edit:				Export:		Autosize:	
	id	id_command	returnDate	addDate	returned	retrieved					
▶	1	14	2014-06-12	NULL	0	0					
*	NULL	NULL	NULL	NULL	NULL	NULL					

FIG. 9.16: Database changes



După ce intră pe cont, operatorul poate accesa lista cu comenzile returnate de către clienţi; la început, produsele sunt în posesia clientului, dar în momentul în care produsele sunt recuperate, operatorul poate modifica baza de date; la început, lista comenzilor returnate arată astfel:

Iuga Delia

Are the order's products back in stock?

Phone number:  
0765432123

Email:  
deliutzzz\_23@yahoo.com

The client wants to return the following products:

Product name	Size	Color	Nr pieces
V TShirt	s	Green	10

(1 of 1) [Previous] [Next]

FIG. 9.17: list

Dacă produsele sunt din nou în stoc, operatorul are posibilitatea de a actualiza numărul de produse şi de a seta comanda ca fiind returnată; acum, clientul îşi poate primi banii înapoi, şi produsele sunt din nou în stoc;

Mai jos este listat codul ce accesează tabelul produselor şi modifică numărul de bucăţi:

```
public void addReturnedProductsPieces (ClientProductDTO
    cp) throws ProductDoesNotExistException {
    Long nrPieces = cp.getNrPieces();
    ProductColorSizeDTO pcs = cp.getProduct();
    List<ProductColorSize> products = em.createNamedQuery(
        ProductColorSize.GET_PRODUCT_COLOR_SIZE).
        setParameter("size", pcs.getSize().getName()).
        setParameter("color", pcs.getProdColor().getColor().
            getName()).setParameter("name", pcs.getProdColor().
            getProduct().getName()).getResultList();
    if (products == null || products.size() == 0) throw new
        ProductDoesNotExistException();
    ProductColorSize product = products.get(0);
    product.setNrOfPieces(nrPieces + product.getNrOfPieces
        ());
    em.merge(product);
}
```

### 9.2.8 Mesajele clienţilor

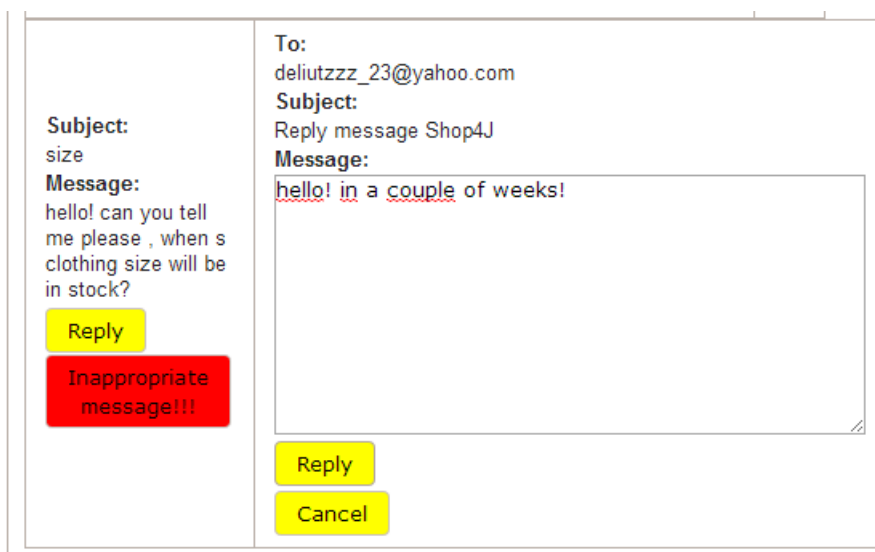
Pentru a-şi arăta părerea sau pentru a pune anumite întrebări despre magazinul şi produsele puse la dispoziţie, clienţii au la îndemână o secţiune unde pot lăsa mesaje; aceste mesaje sunt salvate în baza de date iar apoi sunt vizualizate de către operator.



The screenshot shows a message list interface. On the left, there is a text area with the following content: **Subject:** size, **Message:** hello! can you tell me please , when s clothing size will be in stock?. Below the text area, there are two buttons: a yellow 'Reply' button and a red 'Inappropriate message!!!' button. The entire interface is enclosed in a rectangular frame.

FIG. 9.18: Lista mesajelor

După ce un mesaj a fost primit, operatorul are obligaţia de a se documenta şi apoi de a răspunde clientului în cel mai scurt timp posibil; pentru aceasta, operatorul selectează butonul *Reply* ce deschide un formular pentru completarea răspunsului; evident operatorul se poate răzgândi şi foloseşte butonul *Cancel*; clientul va primi răspunsul pe adresa de e-mail pe care a folosit-o la crearea contului;



The screenshot shows a reply form interface. On the left, there is a text area with the following content: **Subject:** size, **Message:** hello! can you tell me please , when s clothing size will be in stock?. Below the text area, there are two buttons: a yellow 'Reply' button and a red 'Inappropriate message!!!' button. On the right, there is a text area with the following content: **To:** deliutzzz\_23@yahoo.com, **Subject:** Reply message Shop4J, **Message:** hello! in a couple of weeks!. Below the text area, there are two buttons: a yellow 'Reply' button and a yellow 'Cancel' button. The entire interface is enclosed in a rectangular frame.

FIG. 9.19: Rapunsul operatorului

În cazul în care mesajul primit nu este potrivit, operatorul interzice imediat clientului să mai lase şi alte mesaje; astfel se elimină limbajul neadecvat din lista de mesaje;

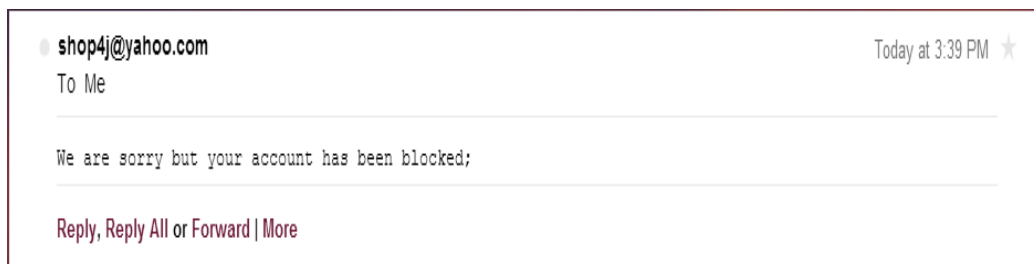


FIG. 9.20: Mail-ul primit de client

Mai jos este prezentat codul Java pe care l-am folosit pentru interzicerea accesului clientului care a lăsat mesajul nepotrivit:

```
public void banUser(User u) throws
    UserNotFoundException{
        if(u== null) throw new UserNotFoundException();
        User user = em.find(User.class, u.getId());
        user.setBan(1);
        em.merge(user);
    }
```

Codul de mai sus are următoarele consecinţe asupra bazei de date:(se observă cum câmpul *ban* este setat la valoarea 1, ceea ce înseamnă că utilizatorul nu mai are acces la cont)

Filter: <input type="text"/> Autosize:							
	id	username	user_password	rolename	email	passwordStatus	ban
	1	delia	delia	client	deliutzzz_23@yahoo.com	0	1

FIG. 9.21: Schimbările din baza de date

### 9.2.9 Chat

Operatorii au de multe ori nevoie de a comunica între ei în legătură cu livrarea unor comenzi, găsirea unui client, primirea unor produse etc; pentru a salva timp preţios, aplicaţia are încorporată un chat disponibil pentru operatori din momentul în care s-au logat până la părăsirea aplicaţiei.

Această funcţionalitate a fost implementată în mare parte în limbajul JavaScript; de asemenea a fost folosită şi biblioteca *jQuery* (facilitează manipularea evenimentelor, crearea de animaţii şi folosirea Ajax; de asemenea funcţionează într-o multitudine de browsere).

Cum se vede în figura alăturată, primul operator iniţiază o conversaţie cu un coleg; operatorul scrie mesajul în input şi apasă tasta *enter*; mesajul este trimis la server care îl trimite colegului de conversaţie; mesajul primit este afişat în inputul celui alt operator;

După cum am spus şi în secţiunea Comet, în spatele chatului se află framework-ul *CometD* ce are la bază modelul Comet ce implementează specificaţiile protocolului *Bayeux*. Comet oferă o bună îndepărtare faţă de modelul de comunicare HTTP, permiţând un stil *push*; Comet defineşte mai multe tehnici care permit serverului să trimită informaţii browserului fără intervenţia clientului; cu ajutorul unei conexiuni HTTP adiţionale, Comet poate facilita comunicare bi-direcţională; [43]



FIG. 9.22: Chat1

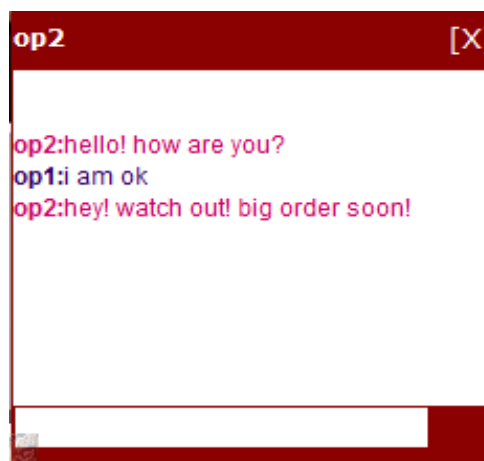


FIG. 9.23: Chat2

```
@Service("chat")
public class ChatService {
    private final ConcurrentMap<String, Map<String, String>>
        _members=new ConcurrentHashMap<String, Map<String,
        String>>();
    @Inject
    private BayeuxServer _bayeux;
    @Session
    private ServerSession _session;
    ...
}
```

Mai jos este prezentat codul *JavaScript* folosit pentru conectarea utilizatorului logat la serverul cometd; după logare utilizatorul subscrie la cele 2 canale *:/chat/demo* şi *members/demo* şi începe să aştepte mesaje de la cele 2 canale;

```
join : function(username) {
    $.cometChat._disconnecting = false;
    $.cometChat.loginUserName = username;
    var cometdURL = location.protocol + "://" + location.host
        + config.contextPath + "/Shop4j/cometd";
    $.cometd.configure({
        url : cometdURL,
        logLevel : 'debug'
    });
    $.cometd.websocketEnabled = false;
    $.cometd.handshake();
}
```

# Capitol 10

## Concluzii

Proiectul de față îmbină 2 mari concepte și anume *cumpărăturile* și *aplicațiile web*.

Cum am menționat în secțiunea Evoluția cumpărăturilor, cumpărăturile sunt în strânsă legătură cu tehnologia: *obiceiul de a cumpăra se schimbă pe măsură ce tehnologia avansează*. Și nici nu ar putea fi altfel; dacă am lăsa ca marile descoperiri să treacă pe lângă noi fără să profităm de beneficiile pe care le aduc și fără să ne adaptăm obiceiurile la schimbările pe care acestea le presupun, atunci nu am cunoaște niciodată evoluția;

Datorită importanței unei aplicații care simulează un magazin online, am ales să implementez o aplicație ușor integrabilă într-un soft folosit în e-commerce;

### **Implementări viitoare:**

1. Analizare review primit de la clienți folosind rețele neurale
2. Modificarea aplicației prin folosirea unei baze de date bazate pe grafuri
3. Folosirea proiectului Apache Hadoop pentru paralelizarea calculelor
4. Data Cubes pentru salvarea datelor

# Bibliography

- [1] Scheiber,E. *Programare distribuită în Java* 147-150
- [2] Linwood, J. ,Minter, D., *Beginning Hibernate* 2010
- [3] Iolu,M. *Sisteme de gestiune a a bazelor de date(MySql,Oracle)* 2012, 5-7
- [4] Keith,M. , Schincariol, M. *Pro JPA 2* 2013
- [5] Mann,K. *JavaServer Faces in action* 2004
- [6] Varaksin, O. , Caliskan,M *PrimeFaces Cookbook* 2013
- [7] Sierra,K. , Bates,B. *Head First Ejb* 2003
- [8] Burke,B. *RESTful Java with JAX-RS 2.0* 2013
- [9] <http://www.businessinsider.com/the-evolution-of-shopping-infographic-2013-6>
- [10] <http://en.wikipedia.org/wiki/E-commerce>
- [11] <http://anamikas.hubpages.com/hub/Online-shopping-sites-benefits>
- [12] [http://en.wikipedia.org/wiki/Online\\_shopping](http://en.wikipedia.org/wiki/Online_shopping)
- [13] <http://www.instantshift.com/2010/03/26/the-history-of-online-shopping-in-nutshell/>
- [14] <http://www.acunetix.com/websitesecurity/web-applications/>
- [15] [http://en.wikipedia.org/wiki/Programming\\_languages\\_used\\_in\\_most\\_popular\\_websites](http://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites)
- [16] <http://www.magicwebsolutions.co.uk/blog/the-benefits-of-web-based-applications.htm>
- [17] <https://www.udemy.com/blog/php-vs-java/>

- [18] <http://java.dzone.com/news/java-versus-php>
- [19] <https://blog.codecentric.de/en/2008/07/comparison-of-java-and-php-for-web-applications/>
- [20] <http://www.roseindia.net/tutorial/java/jdbc/javamvcdesignpattern.html>
- [21] <http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
- [22] [http://tortoisesvn.net/docs/release/TortoiseSVN\\_en/](http://tortoisesvn.net/docs/release/TortoiseSVN_en/)
- [23] [http://en.wikipedia.org/wiki/Client%E2%80%93server\\_model](http://en.wikipedia.org/wiki/Client%E2%80%93server_model)
- [24] <http://www.javaworld.com/article/2077354/learn-java/app-server-web-server-what-s-the-difference.html>
- [25] <http://maven.apache.org/what-is-maven.html>
- [26] [http://en.wikipedia.org/wiki/Apache\\_Maven](http://en.wikipedia.org/wiki/Apache_Maven)
- [27] <http://stackoverflow.com/questions/2356851/database-vs-flat-files>
- [28] [http://www.novell.com/documentation/nw65/web\\_mysql\\_nw/data/aj5bj52.html](http://www.novell.com/documentation/nw65/web_mysql_nw/data/aj5bj52.html)
- [29] [http://www.tutorialspoint.com/hibernate/hibernate\\_architecture.htm](http://www.tutorialspoint.com/hibernate/hibernate_architecture.htm)
- [30] <http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html>
- [31] <http://www.freewebmasterhelp.com/tutorials/xhtml>
- [32] <http://www.htmldog.com/guides/css/beginner/>
- [33] [http://www.tutorialspoint.com/ajax/what\\_is\\_ajax.htm](http://www.tutorialspoint.com/ajax/what_is_ajax.htm)
- [34] [http://www.tutorialspoint.com/javascript/javascript\\_overview.htm](http://www.tutorialspoint.com/javascript/javascript_overview.htm)
- [35] [http://en.wikipedia.org/wiki/Enterprise\\_JavaBeans](http://en.wikipedia.org/wiki/Enterprise_JavaBeans)
- [36] [http://www.tutorialspoint.com/javamail\\_api/](http://www.tutorialspoint.com/javamail_api/)
- [37] <http://itextpdf.com/>



- [38] <http://www.json.org/>
- [39] <https://code.google.com/p/google-gson/>
- [40] <http://docs.oracle.com/javase/tutorial/jaxb/intro/>
- [41] <http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>
- [42] <http://docs.oracle.com/javaee/7/tutorial/doc/jaxrs002.htm>
- [43] <http://www.ibm.com/developerworks/library/wa-cometjava/>
- [44] [http://svn.cometd.org/trunk/bayeux/bayeux.html#toc\\_0](http://svn.cometd.org/trunk/bayeux/bayeux.html#toc_0)
- [45] <http://msdn.microsoft.com/en-us/magazine/ee236638.aspx#id0080022>
- [46] <http://www.mysql.com/products/workbench/>
- [47] <http://www.tutorialspoint.com/jsf/>