

Dezvoltarea aplicațiilor web folosind arhitectura Model-View-Controller (MVC) model 2 și Apache Struts

Date: 07.01.2002

Autor: Sorin Scortan (ss@secure-net.ro)

Compania: SecureNet S.R.L., Craiova, tel.: 051/410555, <http://www.secure-net.ro>

1. Introducere

Dacă sunteți în căutarea unei arhitecturi solide și viabile, care să vă ușureze implementarea aplicațiilor web de o manieră modernă și de viitor, atunci nu veți putea face abstracție de așa-numita arhitectură Model-View-Controller(MVC) .

Una din implementările cele mai reușite ale acestei arhitecturi este framework-ul STRUTS, dezvoltat de către Apache Group ca open source în cadrul proiectului Jakarta.

Pentru a înțelege avantajele oferite de Struts și a putea evalua ce tipuri de proiecte se pretează cel mai bine în a fi abordate și soluționate cu Struts, să urmarim întâi tipul de MVC care a fost implementat în această colecție puternică de clase Java, ca și avantajele și dezavantajele acestei alegeri.

Model-View-Controller (MVC) este un design pattern care leagă eficient interfața cu utilizatorul de modelul de date în programarea orientată pe obiecte. Această arhitectură este larg utilizată în programarea în limbajele Java, C++ sau Smalltalk, permițând reutilizarea codului sursă și reducând astfel durata de dezvoltare a aplicațiilor cu interfețe utilizator.

Arhitectura model-view-controller se constituie din trei componente principale:

- componenta *Model*, reprezentată de structura logică de date a aplicației și clasele de nivel înalt asociate cu ea.

Componenta *Model* nu conține informații despre interfața utilizator.

- componenta *View*, care este o colecție de clase reprezentând interfața cu utilizatorul (toate obiectele pe care utilizatorul le poate vedea pe ecran și cu care poate interacționa, cum ar fi butoane, casete de text, etc.)

- componenta *Controller*, care reprezintă clasele ce realizează comunicarea între clasele din *Model* și cele din *View*

2. MVC Model 1 versus Model 2

Arhitectura MVC s-a dezvoltat în urma nevoilor practicii în două modele ușor diferite, cunoscute sub numele de Modelul 1 și Modelul 2.

Modelul 1 poate fi exemplificat printr-o aplicație simplă de login, a cărei structură este prezentată în fig. 1.

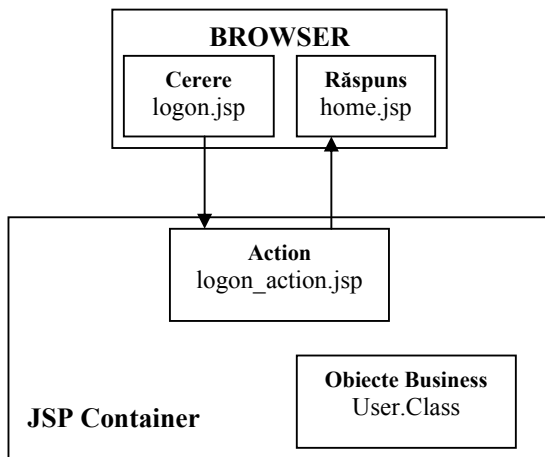


Fig.1: Aplicație login concepută cu MVC Model 1

Logica procesului este secvențială și evoluează din pagina în pagină.

Pagina logon.jsp conține o formă în care se introduc datele de la utilizator.

Pagina logon_action.jsp utilizează o clasă JavaBeans User pentru verificarea datelor și setarea proprietăților Beanului cu valorile pentru user și parolă. Dacă valorile lipsesc sau sunt incorecte, se transferă controlul înapoi paginii logon.jsp. Dacă datele sunt corecte, se transferă controlul paginii home.jsp, care afișează un mesaj de bun venit utilizatorului.

Întreaga logică a aplicației se execută la nivelul paginilor JSP, care folosesc atât cod Java pentru validări cât și cod HTML pentru afișarea rezultatelor.

Paginile logon.jsp și home.jsp constituie componenta de View din model, iar pagina logon_action.jsp constituie componenta de Controller și Model. Clasa User reprezintă un obiect business.

După cum se observă, în paginile JSP se pot amesteca rutine ce țin de logica aplicației (JavaBeans), procesarea la nivel de server (Java și JSP), precum și scripturi la nivel de client (HTML, JavaScript). Containerul JSP este cel care va executa aceste rutine și va transmite rezultatele către una sau alta dintre componentele JSP.

Modelul 1 poate fi utilizat cu succes la dezvoltarea de site-uri simple, avantajul său fiind viteza de dezvoltare a paginilor JSP. Dezavantajele modelului 1 constau în amestecul de componente logice și limbaje la nivelul paginilor JSP, ceea ce face întreținerea lor mai dificilă și conduce la imposibilitatea reutilizării lor pentru alte aplicații.

Modelul 2 reprezintă o structură bazată pe arhitectura model-view-controller (MVC) 2.

Fluxul aplicației este mediat de un Controller central (un servlet sau o colecție de servleturi), care are rolul de a delega cererile HTTP către clasa Java care să le gestioneze.

Componenta Model este constituită dintr-un ansamblu de clase Java care înmagazinează logica aplicației și starea sistemului. După ce logica necesară este executată la nivelul componentelor Model, controlul este transferat înapoi la Controller care îl transmite către paginile JSP care joacă rol de componenta View. Alegerea uneia dintre paginile View care să preia controlul este determinată la nivelul unui fișier de mapări (de regulă un fișier de configurare XML).

În acest fel modelul MVC realizează separarea în componente distincte a logicii aplicației de procesarea la nivel de server și de logica de afișare a rezultatelor. Această separare permite fiecărei componente să poată fi reutilizată și asigură o întreținere mai ușoară a întregii aplicații.

3. Prezentarea arhitecturii Struts

Arhitectura Struts se bazează pe modelul MVC2 și a fost elaborată ca proiect open-source de către Apache Group în cadrul proiectului Jakarta. Actualmente ultima versiune este Struts 1.0 și poate fi încărcată și studiată de pe site-ul <http://jakarta.apache.org/struts>.

Structura arhitecturii Struts este prezentată în fig. 2:

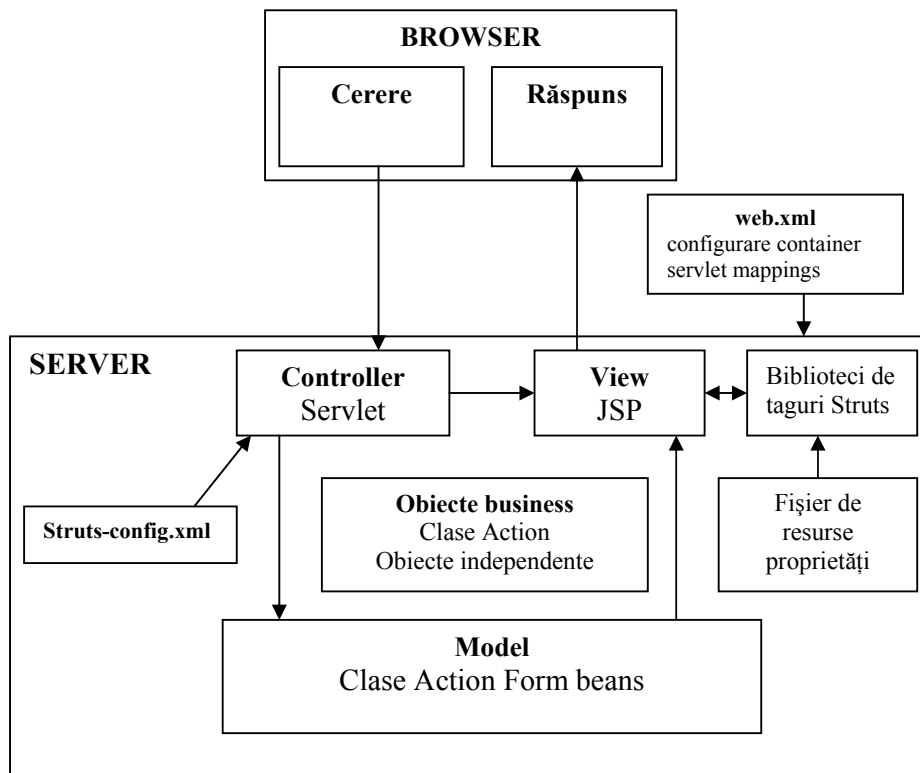


Fig. 2: Arhitectura framework-ului Struts

Componentele arhitecturii Struts, prezentate în ordinea în care participă la proces, sunt:

- web.xml** – fișier de configurare verificat de containerul JSP la pornire. Se specifică controlerele aplicației (ex. Action servlet, database servlet), parametrii de inițializare ai controllerelor (nivelul de debug, dacă se încarcă la startup), maparea unui pattern pentru cererile URI (ex. *.do) la un anumit controler, fișierul JSP de start și locația bibliotecilor de taguri JSP Struts precum și a bibliotecilor de taguri create de utilizator
- Cererea** – cererea HTTP de la utilizator. Se cere afișarea unei anumite pagini sau se transmite o formă cu date de intrare. Această cerere este interceptată de controler

- c. **Controlerul** - este procesorul central al arhitecturii și este responsabil pentru rutarea cererilor HTTP la obiectele corespunzătoare din arhitectură. Controlerul este un servlet (ActionServlet). Pot exista mai multe servleturi la nivel de aplicație (ex. DatabaseServlet pentru inițializarea unui connection pool și conectarea la o bază de date). Când controlul este inițializat, se citește fișierul de configurare struts-config.xml ce specifică acțiunile mapate la nivelul aplicației. Controlerul utilizează aceste mapări pentru a determina unde să transfere cererea HTTP. Controlul poate fi transferat unei clase Action care să execute logica aplicației, sau înaintat unei componente JSP pentru afișarea în browser.
- d. **Struts-config.xml** – fișier de configurare care păstrează toate acțiunile mapate la nivelul aplicației. Se asigură modularizarea aplicației și prevenirea apelului direct al unei componente din browser. Maparea acțiunilor este păstrată în memorie la nivelul unor clase ActionMapping. O clasă ActionMapping definește o cale URI care corespunde cererii HTTP sosite din browser și specifică numele clasei Action care va executa această cerere. De asemenea se specifică dacă validarea datelor sosite împreună cu cererea se face la nivelul clasei ActionForm sau al clasei Action. Se mai specifică și unde se transferă controlul în cazul în care execuția logicii se realizează cu succes sau în cazul în care apare o eroare la validare.
- e. **Model** – modelul este reprezentat de un obiect ce preia răspunsul de la utilizator și îl păstrează pe durata procesului. De regula modelul este constituit din clase ActionForm și clase Action. Clasele ActionForm au forma unor JavaBeans cu metode set() și get() care înmagazinează pe durata aplicației datele introduse de utilizator în formele HTML. De obicei există câte o clasă ActionForm pentru fiecare formă de intrare. Validarea datelor poate fi executată și la nivelul claselor ActionForm prin suprascierea metodei validate(). Clasele ActionForm mai conțin și metoda reset() pentru reinițializarea proprietăților în cazul în care utilizatorul apasă butonul Reset dintr-o formă.

În cazul în care un anumit proces nu necesită utilizarea unui obiect model, controlerul va transmite controlul direct unui obiect View.

Clasele Action vor executa logica aplicației. Ele pot apela proprietățile unor obiecte de business (independente de implementare), pot actualiza sau citi date din baza de date și pot executa operații de validare. Eventualele erori găsite pe durata validării sunt scrise sub forma unor obiecte ActionError într-o colecție de erori ActionErrors, urmând a fi afișate de componenta View folosind un tag Struts <html:error>

La final se scriu în contextul containerului toate obiectele necesare pentru afișarea rezultatelor, iar controlul este transferat către controler care îl redirecționează către o componenta View specificata în struts-config.xml

- f. **Obiecte business** – obiecte ce înmagazinează datele aplicației. De regulă sunt obiecte care extind clasa Java.lang.object și sunt independente de clasele Struts sau HTTP Servlets, avantajul fiind ca pot fi reutilizate și în alte aplicații. Ele sunt apelate de clasele Action sau din paginile JSP de ieșire.
- g. **View** – obiectele View sunt pagini JSP care afișează în browser rezultatele cererii. Paginile JSP conțin taguri Struts ca și taguri definite de utilizator. Ele nu conțin cod Java ca în modelul 1, astfel încât paginile JSP pot fi implementate de un web designer fără cunoștințe de Java. Mesajele afișate în browser pot fi internaționalizate cu ajutorul fișierului de resurse
- h. **Biblioteci de taguri Struts** – sunt componente Struts utilizate de obiectele View pentru afișarea logicii aplicației. Ele mapează toate tagurile HTML precum și codul Java existent în paginile JSP de model 1.
- i. **Fișiere de resurse proprietăți** – sunt fișiere de proprietăți care păstrează mesajele afișate în obiectele View, utile în internaționalizarea aplicației web.
- j. **Răspunsul** – reprezintă pagina finală pe care utilizatorul dorește s-o vadă, de regulă un obiect View ca o pagina JSP.

Structura aplicației de login folosind arhitectura Struts ar arăta astfel:

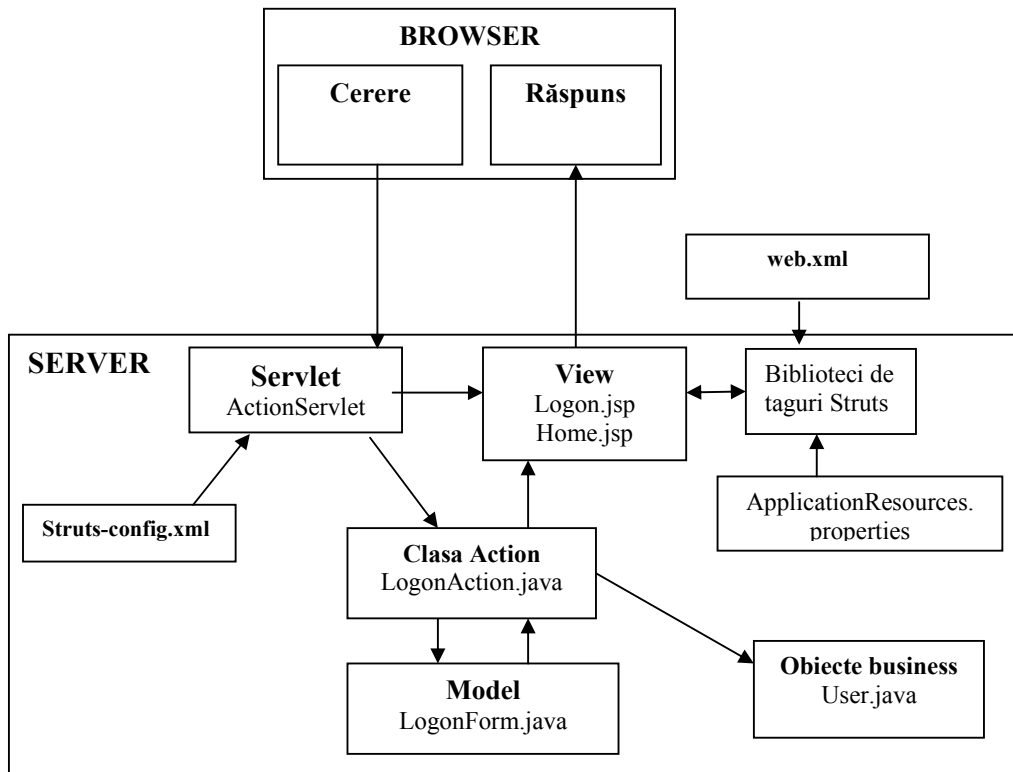


Fig. 3: Aplicația login folosind framework-ul Struts

În caseta următoare se prezintă sursele comentate ale componentelor aplicației.

Logon.jsp

```
<%@ page language="java" %>
<!--initializare librarie de taguri Struts Bean-->
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<!--initializare librarie de taguri Struts HTML-->
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>

<html:html locale="true">
<head>
<title>Aplicatie login cu Struts</title>
<html:base/>
</head>
<body bgcolor="white">
<!--afisare eventuale erori-->
<html:errors/>
<p align="center">Aplicatie de login cu Struts</p>
<!--forma de login-->
<html:form action="jsp/logon.do" focus="user">
<table>
<tr>
<td>
<td> <html:text property="user" size="20" maxlength="20"/></td>
</tr>
</table>
```

```

<tr>
<td colspan="2">
<!--buton de submit-->
<html:submit property="submit" value="Login"/>
</input>
</td>
</tr>
</table>
</html:form>
</body>
</html:html>

```

Not•: logon.do nu exist• nicaieri, dar fisierul web.xml con•ine informa•ii de configurare utilizate de containerul JSP ca s• mapeze toate cererile de fi•iere care se termin• cu .do spre ActionServlet.

În web.xml exist• urm•toarele linii de cod:

```

<!-- Standard Action Servlet Mapping -->
<servlet-mapping>
<servlet-name>action</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>

```

ActionServlet va citi c•uta în fi•ierul struts-config.xml o ac•iune care se mapeaz• c•tre logon.jsp. Fragmentul urm•tor de cod din struts-config.xml comunic• servletului s• utilizeze clasa Logonaction pentru a procesa forma:

```

<action-mappings>
<!-- logon action -->
<action path="/jsp/logon.jsp"
type="ro.securenet.struts.exemplu.LogonAction"
name="exemplu">
<forward name="home" path="/jsp/home.jsp"/>
<forward name="logon" path="/jsp/logon.jsp"/>
</action>
</action-mappings>

```

Home.jsp

```

<%@page contentType="text/html"%>
<jsp:useBean id="User" scope="session"
class="ro.securenet.struts.exemplu.User" />
<!-- verificare daca userul e activ -->
<% if (User.isActive() == false){%>
<jsp:forward page='logon.jsp' />
<%}%>

<html>
<head><title>Home Page</title></head>
<body>
<p align="center"> Exemplu MVC Model 2 cu Struts</p>
<p> Bine ai venit:
<jsp:getProperty name="User" property="userId" />
</p>
</body>
</html>

```

User.java - o clasa de tip JavaBean

```

package ro.securenet.struts.exemplu;

public class User extends Object {
/** creeaz• un obiect User */
public User() {}

//userid: p•steaz• un id unic per user
private String userId = null;
public String getUserId(){ return userId; }
public void setUserId(String aUser) { userId = aUser; }

```

```

//active=true dac• userul curent e activ
private boolean active = false;
public boolean isActive() { return active; }
public void setActive(boolean is_active) { active = is_active; }
}

```

LogonAction.java

```

package ro.securenet.struts.exemplu;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.struts.action.*;
import org.apache.struts.action.ActionForm;
import ro.securenet.struts.*;

/**
 * Clasa Action pentru initializare logica de login.
 */
public class LogonAction extends Action
{
    public ActionForward perform(ActionMapping mapping,
    ActionForm form, HttpServletRequest request,
    HttpServletResponse response)
    {
        //LogonForm este clasa ActionForm asociata cu aceasta cerere
        if (form instanceof LogonForm) {
            String ls_action = "logon";
            LogonForm l_data = (LogonForm) form;

            // citeste user id
            String ls_user = l_data.getUser();
            User l_new_user = new User(ls_user);

            // Plaseaza obiectul User in sesiune pentru utilizare ulterioara
            HttpSession the_session = request.getSession() ;
            the_session.setAttribute("User", l_new_user);

            /* Determina clasa View destinatie */
            if (l_new_user.isActive()) ls_action = "home";

            return mapping.findForward(ls_action);
        }
        return null;
    }
}

```

Logonform.java

```

package ro.securenet.struts.exemplu;

import java.beans.*;
import org.apache.struts.action.*;
import ro.securenet.struts.*;

/**
 * Clasa ActionForm
 */
public class Logonform extends ActionForm implements java.io.Serializable {

    public Logonform () {}

    private String user;

    public void setUser(String as_user)
    {user = as_user;}
}

```

```
public String getUser()
{return(user);}
}
}
```

Not•: clasa LogonForm de tip ActionForm este specificat• •i în struts-config.xml în tagul:

```
<form-beans>
<form-bean name="exemplu" type="ro.securenet.struts.exemplu.LogonForm"/>
</form-beans>
```

ActionServlet.java - este cel furnizat de arhitectura Struts •i este specificat in **web.xml** în tagurile:

```
<servlet-name>action</servlet-name>
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
```

Tot în **web.xml** se specific• loca•ia fi•ierului de configurare struts-config.xml •i eventual a unui fi•ier de mesaje la nivel de aplica•ie, ApplicationResources.properties:

```
<init-param>
<param-name>application</param-name>
<param-value>ApplicationResources</param-value>
</init-param>
<init-param>
<param-name>config</param-name>
<param-value>/WEB-INF/struts-config.xml</param-value>
</init-param>
```

Alte arhitecturi înrudite:

Turbine – arhitectură MVC model 2 care permite dezvoltarea de aplicații web securizate. Dispune de o serie de servicii specializate (connection pool, scheduler jobs, caching, internaționalizare, integrare Log4J, transformări XSLT, XML-RPC, suport pentru WebMacro și Velocity, mapare baze de date) și este dezvoltat de Apache ca proiect open-source.

Mai multe informații la <http://jakarta.apache.org/turbine>

Expresso – arhitectură MVC model 2 pentru dezvoltare de aplicații web, concepută de firma JCorporate (<http://www.jcorporate.com>), integrează Struts din versiunea 4.0 și dispune de servicii de securizare și mapare baze de date, scheduler jobs, connection pool, caching, integrare Log4J, notificare evenimente, librării de taguri JSP și altele.

Avantajele arhitecturii Struts:

- Struts permite modularizarea componentelor aplicației, de aici rezultând o dezvoltare mai rapidă a componentelor și specializarea lor pentru realizarea anumitor funcții, cu avantaje în reutilizarea componentelor și în alte aplicații.
- starea variabilelor introduse în formele HTML este păstrată la nivel de server în beanuri ActionForm, fiind ușor de procesat și de păstrat de-a lungul aplicației. Controlerul verifică existența unei instanțe a beanului de clasa respectivă, iar dacă nu există se creează imediat una și se adaugă la sesiune. Apoi controlerul apelează metoda set() a fiecărui parametru din cererea HTTP ce se potrivește cu numele unei proprietăți în bean.
- Există posibilitatea execuției logicii aplicației în mai mulți pași de către mai multe clase Action care cooperează între ele și care păstrează informațiile importante la nivelul contextului container al aplicației. Ele pot apela proprietăți ale unor obiecte de business care păstrează starea aplicației sau pot apela obiecte care mapează datele din baza de date.
- Struts are un mecanism propriu de culegere a erorilor în obiecte ActionError și de afișare a lor într-o forma internaționalizată. Operațiunile de validare a datelor introduse de utilizator se realizează la nivelul componentelor model (ActionForm și Action) și nu la nivelul componentelor View.
- Obiectele business create în modelul 1 pot fi utilizate și cu arhitectura Struts.

- Struts posedă un mecanism de log la nivel de context container, cu specificarea nivelelor de log și a mesajelor ce se vor înmagazina într-un fișier de log.
- Struts conține suport pentru baze de date și are o clasa ce creează un connection pool, accesat prin JDBC. În fișierul struts-config.xml se pot defini mai multe surse de date.

Dezavantajul principal al arhitecturii Struts este acela că este destul de complex și necesită o perioadă de învățare și de acomodare atât cu terminologia specifică modelului MVC 2, cât și cu funcționalitatea componentelor și comunicarea dintre ele.

4. Concluzii

Aspectul cel mai important al arhitecturii Struts este soliditatea designului său. Arhitectura integrată Struts permite dezvoltarea mai rapidă a aplicațiilor, internaționalizarea lor și separarea ca și paralelizarea sarcinilor de programare între programatorii Java și web designeri. În plus, dezvoltatorii se pot concentra pe dezvoltarea logicii aplicației și pe prezentarea datelor către utilizator, lăsând comunicarea dintre componente pe seama arhitecturii Struts.

5. Referințe

- Arhitectura Struts din proiectul Apache Jakarta - <http://jakarta.apache.org/struts>
- Arhiva Struts - <http://www.mail-archive.com/struts-user@jakarta.apache.org/>
- Husted Dot Com - <http://www.husted.com/struts/>
- Struts, an Open-Source MVC Implementation (articol despre Struts) - <http://www-106.ibm.com/developerworks/library/j-struts/index.html?open&l=jls>

6. Prezentare autor

Sorin Scortan lucrează ca inginer software la SecureNet S.R.L., în Craiova. El are o experiență de 5 ani în dezvoltarea de aplicații client/server și web folosind Java, Visual Basic și sisteme de baze de date relaționale.

