

UNIVERSITATEA ALEXANDRU IOAN CUZA IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

FII Project Keeper

propusă de

Pantelemon Victor-Ștefan

Sesiunea: *iulie, 2019*

Coordonator științific

Lect. dr. Cristian Frăsinaru

UNIVERSITATEA ALEXANDRU IOAN CUZA IAȘI

FACULTATEA DE INFORMATICĂ

Fii Project Keeper

Pantelemon Victor-Ștefan

Sesiunea: *iulie, 2019*

Coordonator științific

Lect. dr. Cristian Frăsinaru

DECLARAȚIE PRIVIND ORIGINALITATE ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul “*Titlul complet al lucrării*” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imagini etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, *data*

Absolvent *Prenume Nume*

(semnătura în original)

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul “*Titlul complet al lucrării*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea Alexandru Ioan Cuza Iași să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent *Prenume Nume*

(semnătura în original)

Acord privind proprietatea dreptului de autor

Facultatea de Informatică este de acord ca drepturile de autor asupra programele-calculator, format executabil și sursă, să aparțină autorului prezentei lucrări, *Prenume Nume*.

Încheierea acestui acord este necesară din următoarele motive:

[Se explică de ce este necesar un acord, se descriu originile resurselor utilizate în realizarea produsului-program (personal, tehnologii, fonduri) și aportul adus de fiecare resursă.]

Iași, *data*

Decan *Prenume Nume*

Absolvent *Prenume Nume*

(semnătura în original)

(semnătura în original)

Cuprins

1. Introducere.....	7
2. Contributii.....	8
3. Utilizarea aplicației.....	9
4. Arhitectura.....	10
5. Java Server Faces.....	12
6. PrimeFaces.....	17
7. Componente personalizate.....	18
8. Internaționalizare.....	20
9. Securitate.....	22
10. Baza de date.....	25
11. Validarea datelor.....	28
12. Preluarea și afișarea datelor.....	29
13. Manipularea fișierelor de dimensiuni mari.....	31
14. Antiplagiere.....	34
15. Manual de utilizare.....	40
16. Concluzii.....	50
17. Bibliografie.....	51

1.Introducere

Problema pe care o rezolvă această licență este lipsa unui sistem centralizat de gestiune a licențelor și a proiectelor din facultate.

În prezent există aplicații precum *GitHub*, *Dropbox*, *Google Drive* etc. care rezolva într-o oarecare măsură această problema.

Principalul scop al acestui proiect este de a oferi atât studenților cât și profesorilor o aplicație centralizată care să faciliteze stocarea diverselor proiecte din facultate cât și disponibilitatea lor.

Față de aplicațiile prezentate anterior acest proiect aduce în plus o modalitate de asociere a anumitor proiecte în cadrul unei teme propuse sau în cadrul rezolvării unei probleme similare, oferirea unei descrieri pe scurt atât a problemei rezolvate cât și a modului de implementare, posibilitatea de a oferi o prezentare a soluției cât și soluția în sine, un sistem de management al accesului la anumite resurse, eventual niște statistici făcute pe baza modului în care utilizatorii interacționează cu aplicația și posibilitatea în timp a utilizării unui sistem de detectare a plagierii.

Principalele tematici abordate în această lucrare sunt "Dezvoltarea aplicațiilor web", specific utilizarea unui framework care să faciliteze legătură entităților din frontend cu cele din backend, utilizarea și integrarea în aplicație a unor componente web, crearea și reutilizarea propriilor componente web, internaționalizarea și localizarea, Implementarea unui sistem de control al accesului pe baza de roluri, managementul sesiunii, utilizarea optimă a resurselor, manipularea șirurilor de octeti, utilizarea unui sistem persistent de stocare, analiza datelor și generarea unor statistici și scheme, analiza codului sursă și parsarea acestuia, generarea unei structuri abstracte pe baza tokenilor identificați la parsare eventual compararea structurilor generate și asocierea elementelor similare.

2.Contribuții

Contributile absolventului în dezvoltarea acestei lucrări au fost cercetarea tehnologiilor utilizate, planificarea arhitecturii aplicației, planificarea modului în care utilizatorii interacționează cu aplicația, stabilirea unui plan de lucru în scopul dezvoltării acestei lucrări, implementarea soluției, documentarea în privința tehnicilor de programare, căutarea și alegerea unei soluții optime, testarea soluției, semnalarea în mediul open source a unor probleme existente în unele software utilizate.

Tehnologii:

Tehnologiile utilizate în acest proiect sunt Java Server Faces(JSF), PrimeFaces și PostgreSQL.

3.Utilizarea aplicației:

Utilizatorul aplicației în calitate de profesor poate să înscrie în aplicație o temă de proiect (exemplu “Licență 2019”).

Înscrierea temei cuprinde numele temei de proiect, anii de studiu pentru care este adresată (exemplu toți studenții din anul 3), un termen limita până când să fie finalizat proiectul și o secțiune de detalii suplimentare unde se va menționa cerința proiectului eventual limbaje recomandate și materia cu care poate să fie asociat proiectul.

Profesorul are control deplin asupra temei de proiect, el poate să vizualizeze toate proiectele încărcate de studenți.

Profesorul nu poate șterge o tema de proiect și nici nu poate șterge sau suprascrie un proiect care nu îi aparține.

Deasemenea, profesorul poate să încarce propriul proiect, poate să vizualizeze toate temele propuse și poate să închidă și să redeschidă orice tema de proiect.

Utilizatorul în calitate de student poate să vizualizeze toate temele în care este implicat sau în care a luat parte. El poate să încare un proiect și să-și vizualizeze propriul proiect. El nu poate vedea toate temele de proiect și nici temele celorlalți.

O temă de proiect închisă poate să fie doar vizualizată, alte proiecte nu se mai pot încărca până tema de proiect nu este redeschisă.

4.Arhitectura

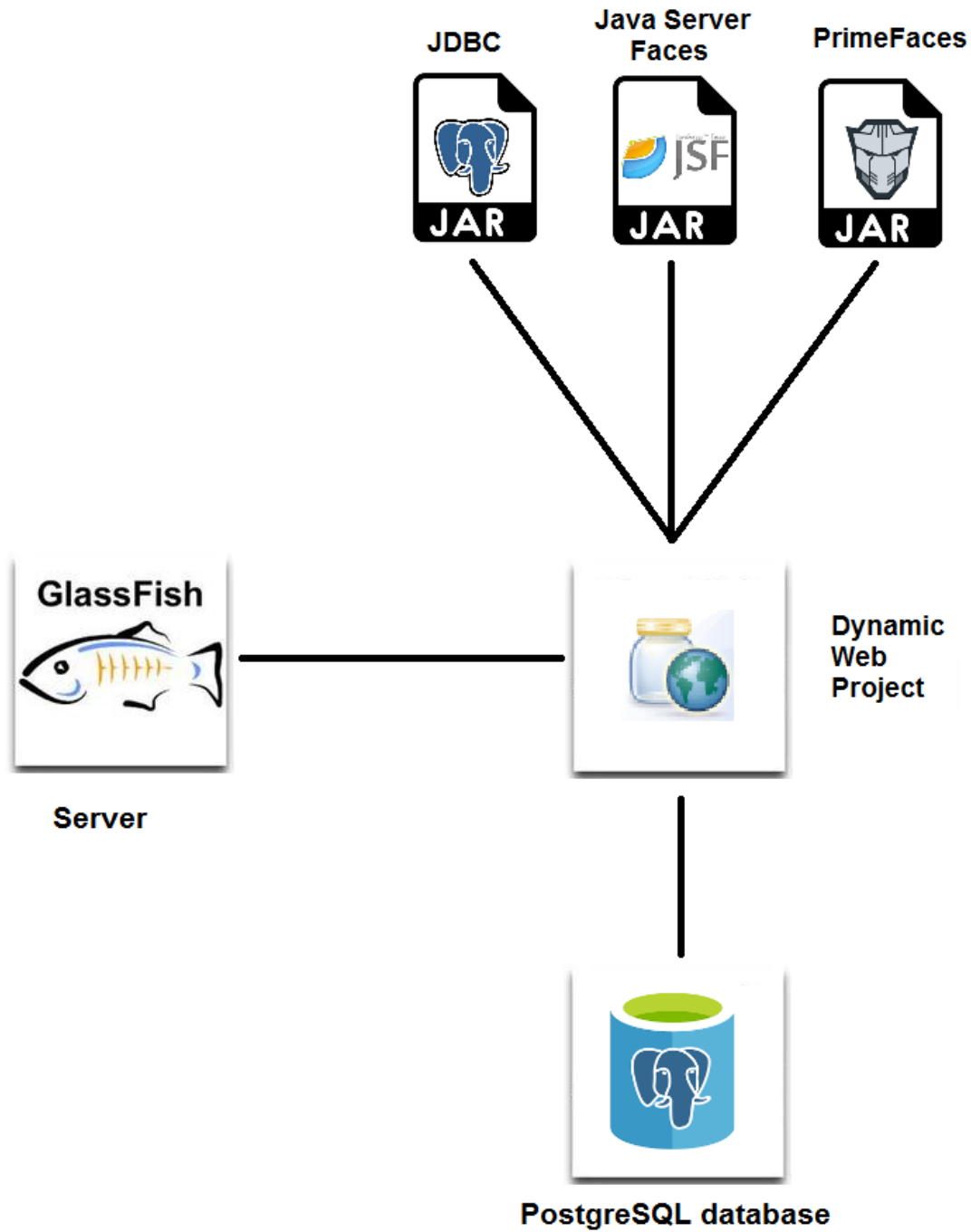
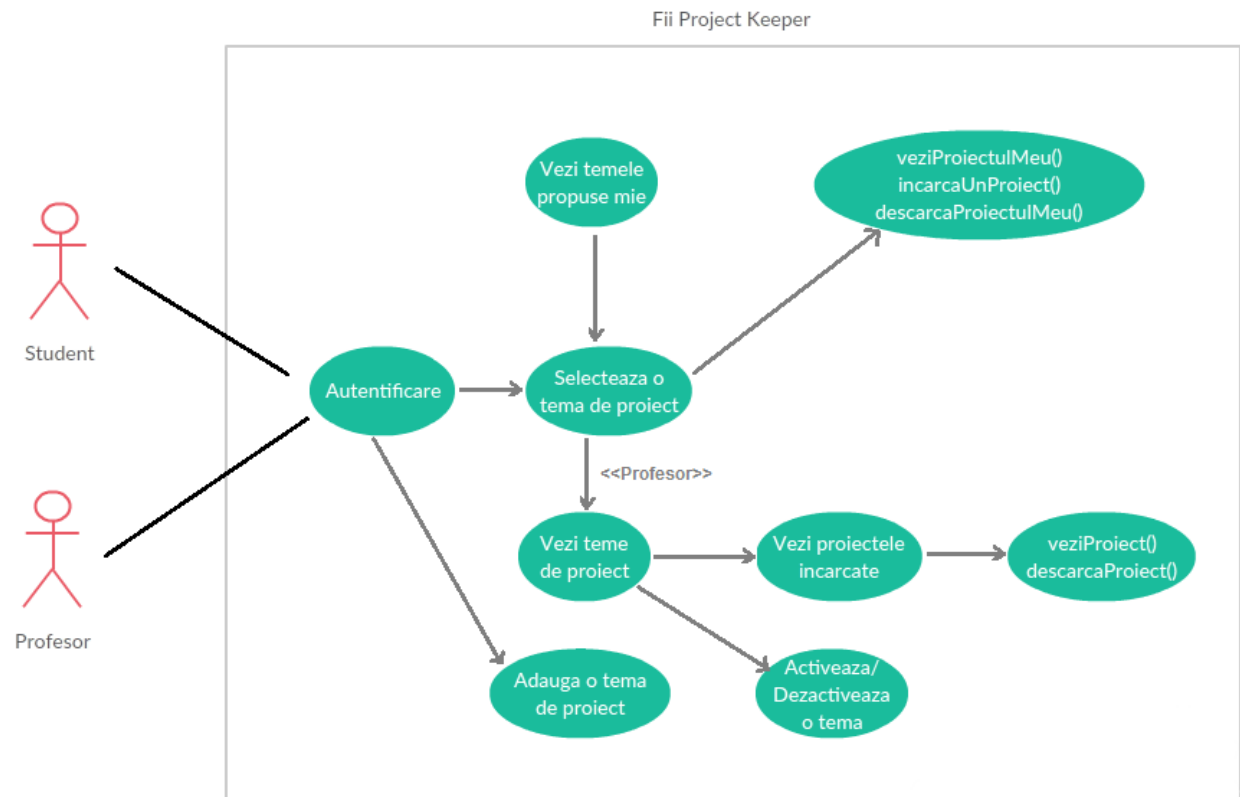


Diagrama Use Case:



5.Java Server Faces(JSF)

JavaServer Faces (JSF) este o specificație Java pentru construirea de interfețe utilizator bazate pe componente pentru aplicații web și a fost formalizată ca standard prin intermediul Java Community Process, care face parte din Platforma Java Enterprise Edition.

Este, de asemenea, un cadru web MVC care simplifică construirea de interfețe utilizator (UI) pentru aplicațiile bazate pe server, utilizând componente UI reutilizabile într-o pagină.

JSF 2 folosește Facelets ca sistem default de template-uri. Pot fi utilizate și alte tehnologii de vizualizare, cum ar fi XUL sau simplu, Java.

În schimb, JSF 1.x utilizează JavaServer Pages(JSP) ca sistem default de template-uri.

JSF în Fii Project Keeper

JSF este folosit în proiect pentru a genera în mod dinamic cod HTML și Javascript.

Acesta folosește componente scrise în cod XHTML care este procesat la runtime și randează codul HTML respectiv.

XHTML

```
<h:form enctype="multipart/form-data" id="panel">
  <p:growl id="growl" life="10000" />

  <h:panelGrid columns="1" width="100%" cellpadding="10px">

    <h:panelGroup>
      <div class="white-roundEdges-div">
        <dl>
          <dt>Descriere</dt>
          <dd>
            <p:inputTextarea rows="4" cols="40" maxlength="255"
              counterTemplate="{0} characters remaining." autoResize="false"
              value="#{projectUploadView.description}"
              <f:validateRegex pattern="^(?!'.)*$" /></f:validateRegex>
            </p:inputTextarea>
          </dd>
        </dl>
      </div>
    </h:panelGroup>
    <h:commandButton class="btn btn-primary" value="#{msg['submit']}"
      icon="ui-icon-check" type="submit"
      action="#{projectUploadView.upload}" update="panel" />

  </h:panelGrid>

</h:form>
```

HTML

```
<form id="panel" name="panel" method="post" action="/Fii Project Keeper/faces/upload.xhtml" enctype="multipart/form-data">
  <input type="hidden" name="panel" value="panel">
  <span id="panel:growl"></span>
  <table width="100%" cellpadding="10px">
    <tbody>
      <tr>
        <td>
          <div class="white-roundEdges-div">
            <dl>
              <dt>Descriere</dt>
              <dd>
                <textarea id="panel:j_idt21" class="ui-inputfield ui-inputtextare ui-widget ui-state-default ui-corner-all"
                  name="panel:j_idt21" cols="40" rows="4" maxlength="255" role="textbox" aria-disabled="false" aria-readonly="false" aria-
                    multiline="true"></textarea> <event>
                </dd>
              </dl>
            </div>
          </td>
        </tr>
      <tr>
        <td>
          <input class="btn btn-primary" type="submit" name="panel:j_idt31" value="Incarca">
          </td>
        </tr>
      </tbody>
    </table>
```

JSF elimina complet conceptul de API acesta făcând automat legătura între entitățile din pagină cu cele din backend. Pentru a putea asocia un element din pagină cu o clasă din Java aceasta trebuie să aibă adnotarea “@ManagedBean” iar proprietatea sau funcția să fie publică(sau proprietatea să aibă getter și setter public). Numele clasei recunoscute în frontend se poate specifica astfel “@ManagedBean(name=”MyClass”)”. Implicit se folosește numele clasei începând cu litera mică.

XHTML

```
<p:inputTextarea rows="4" cols="40" maxlength="255"
  counterTemplate="{0} characters remaining." autoResize="false"
  value="#{projectUploadView.description}">
  <f:validateRegex pattern="^((?!').)*$" /></f:validateRegex>
</p:inputTextarea>
<h:commandButton class="btn btn-primary" value="#{msg['submit']}"
  icon="ui-icon-check" type="submit"
  action="#{projectUploadView.upload}" update="panel" />
```

Java

```
@ManagedBean
public class ProjectUploadView {

    private String description;
    private Part presentation;
    private Part file;

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public boolean upload() {
        if(validateFiles())
        {
            //display();

            DataAccessAPI api=new DataAccessAPI();
            Project project=getAsProject();
            if(api.addProject(project))
            {
                Security.getSession().viewProject();
                return true;
            }
        }
    }
}
```

Managementul sesiunii in JSF

La randarea paginii fiecare clasă a cărei metode sau proprietate este menționată în codul XHTML este instantiată în mod automat.

Pentru persistența instanțelor se folosește adnotarea "@SessionScoped". Instanțele claselor sunt distruse odată cu sesiunea.

Randarea conținutului dinamic

Dinamicitatea conținutului este dată prin folosirea valorilor campurilor din clase.

```
<a class="nav-link" href="#">Logged as #{user.username}</a>
```

Folosirea structurilor condiționale pentru afișarea diverselor elemente în funcție de rol sau randarea unor butoane pentru diverse operații.

```
<c:if test="#{user.type=='Student'}">
  <div class="row text-center">

    <div class="col-lg-3 col-md-6 mb-4" style="left:38%; ">
      <div class="card" >
        
        <div class="card-body">
          <h4 class="card-title"><h:outputText value = "Vezi proiectele" />.</h4>
        </div>
        <div class="card-footer">
          <a href="repositories.xhtml" class="btn btn-primary"><h:outputText value = "Proiecte" /></a>
        </div>
      </div>
    </div>
  </div>
</c:if>
```

Folosirea structurilor repetitive pentru afișarea entităților din baza de date.

```
<p:dataGrid var="proiect" value="#{repositoriesView.proiecte}" columns="3" layout="grid" id="proiecte" styleClass="ui-datagrid-no-border" >

  <div class="card" style="margin:10px; text-align:center; background-color:#0255ff1a; " >
    <h3><h:outputText value="#{proiect.title}" /></h3>
    <div style="margin-left:10px;background:#eae0980; height:100%;">
      <dl>
        <dt>Materie</dt>
        <dd><h:outputText value="#{proiect.subject}" /></dd>
        <dt>Data crearii</dt>
        <dd><h:outputText value="#{proiect.getFormattedDate()}" /></dd>
        <dt>Deadline</dt>
        <dd><h:outputText value="#{proiect.data}" /></dd>
      </dl>
    </div>

    <div class="card-footer trg" style="background-color:#007bff75;">
      <!-- <a href="#{repositoriesView.proiecte.indexOf(proiect)}" class="btn btn-primary">Open</a> -->
      <p:commandLink value="Open" actionListener="#{repositoriesView.viewProject(repositoriesView.proiecte.indexOf(proiect))}" />

    </div>

  </div>

</p:dataGrid>
```


6.PrimeFaces

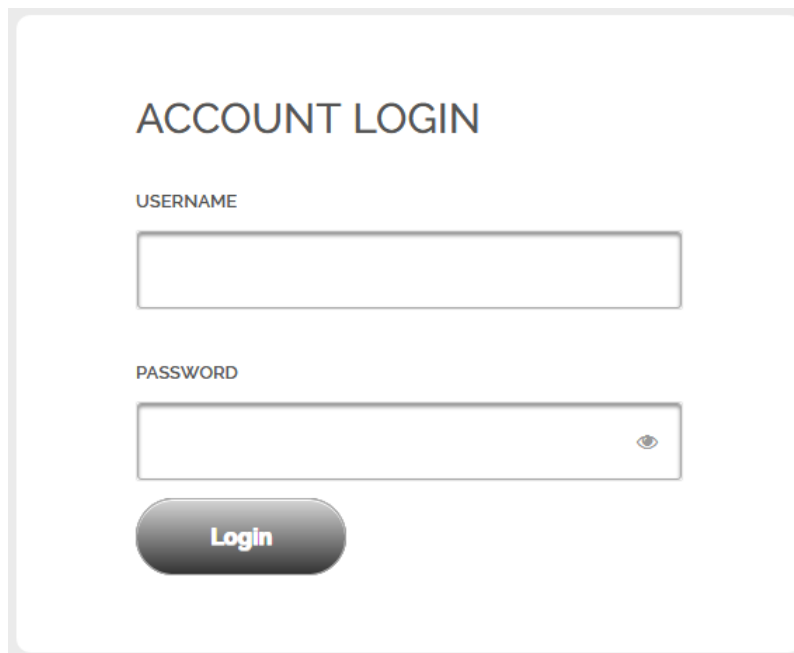
PrimeFaces este o bibliotecă de interfață cu utilizatorul open source (UI) pentru aplicațiile JavaServer Faces, creată de compania turcă PrimeTek Informatics.

Dezvoltarea inițială a PrimeFaces a început la sfârșitul anului 2008. Predecesorul PrimeFaces este biblioteca YUI4JSF, un set de componente JSF bazate pe biblioteca JavaScript YUI.

YUI4JSF a fost anulat în favoarea PrimeFaces la începutul anului 2009.

De la lansarea sa, PrimeFaces a fost puternic sprijinit de Oracle, în special în lumea NetBeans.

Multe elemente vizuale utilizate în proiect precum butoane, zone de completat, tabele dar și funcții în Java aparțin sunt preluate de pe pagină oficială PrimeFaces.



ACCOUNT LOGIN

USERNAME

PASSWORD

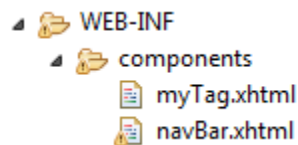
Login

7.Componente Personalizate

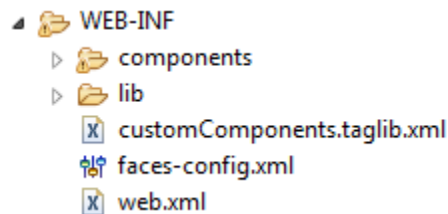
Pentru evitarea repetabilitatii și rezolvarea problemei modificării unei componente comune pe mai multe pagini s-a recurs la definirea și reutilizarea unei componente personalizate, în cazul de față bara de navigație.



Pentru construirea unei astfel de componente este nevoie ca aceasta să fie definită într-un fișier xhtml separat.



Fișierele trebuiesc menționate într-un document xml care trebuie referit în web.xml.



customComponents.taglib.xml

```
<facelet-taglib>
  <namespace>http://tutorialspoint.com/facelets</namespace>

  <tag>
    <tag-name>myTag</tag-name>
    <source>components/myTag.xhtml</source>
  </tag>

  <tag>
    <tag-name>navBar</tag-name>
    <source>components/navBar.xhtml</source>
  </tag>
</facelet-taglib>
```

web.xml

```
<context-param>  
  <param-name>javax.faces.FACELETS_LIBRARIES</param-name>  
  <param-value>/WEB-INF/customComponents.taglib.xml</param-value>  
</context-param>
```

Spațiul de nume al componentei trebuie să fie declarat în toate paginile unde aceasta urmează să fie folosită.

```
<html lang="en"  
xmlns="http://www.w3.org/1999/xhtml"  
  xmlns:h="http://java.sun.com/jsf/html"  
  xmlns:f="http://java.sun.com/jsf/core"  
  xmlns:p="http://primefaces.org/ui"  
  xmlns:c="http://java.sun.com/jsp/jstl/core"  
  xmlns:ex="http://tutorialspoint.com/facelets">
```

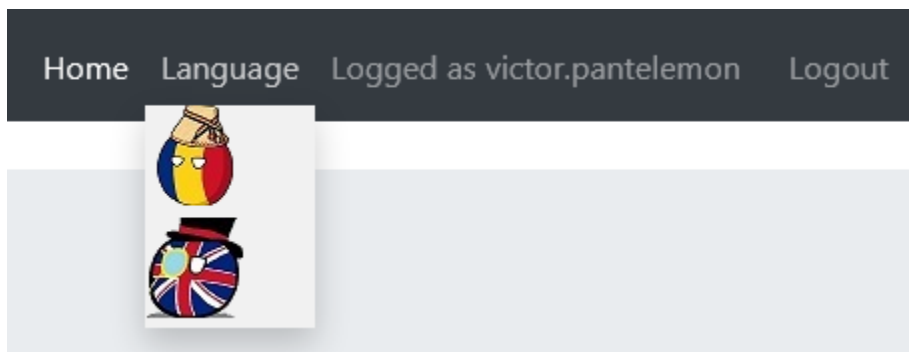
```
<!-- Navigation -->
```

```
<ex:navBar/>
```

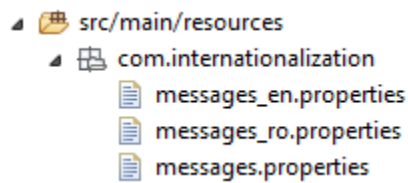
Un lucru foarte important este că aspectul și modul de funcționare a componentei să fie independent de pagina pe care se află. Motiv pentru care este recomandat ca stilul componentei să fie definit separat în componentă.

8. Internaționalizare

Interfața web este disponibilă în 2 limbi și anume română engleză, limba se poate selecta din bara de navigație din secțiunea "Language".



Implementarea internaționalizării se face prin definirea unor dicționare care conțin cuvântul cheie și traducerea sa în limba dorită.



messages_ro.properties

```
welcome=Bine ai venit  
load_a_project=Incarca un proiect  
upload=Incarca  
download_a_project=Descarca un proiect  
download=Descarca  
settings=Setari
```

Fișierele sunt apoi declarate în fișierul faces-config.xml.

```
<application>
  <message-bundle>main.resources.com.internationalization.messages</message-bundle>

  <locale-config>
    <default-locale>ro</default-locale>
    <supported-locale>en</supported-locale>
  </locale-config>

  <resource-bundle>
    <base-name>com.internationalization.messages</base-name>
    <var>msg</var>
  </resource-bundle>

</application>
```

Cuvintele cheie sunt referite în paginile XHTML sub formă de dicționar. Valoarea lor este dată de limba predefinită.

```
<center>
  <p:outputLabel value="#{msg['load_a_project']}" style="font-size:50px"/>
</center>
```

Pentru schimbarea limbii se completează un formular disponibil în bara de navigație, acesta apelează o funcție disponibilă în clasa "Language.java". Această clasa este statică la nivel de sesiune.

```
<h:form>
<div class="dropdown-content">
  <h:commandButton action="#{language.changeLanguage('ro')}}"
    value="Romanian" image="resources/img/ro.png" />

  <h:commandButton action="#{language.changeLanguage('en')}}"
    value="English" image="resources/img/en.png" />

</div>
</h:form>
```

Language.java

```
public void changeLanguage(String language) {
    System.out.println("Changed language");
    locale = new Locale(language);
    FacesContext.getCurrentInstance().getViewRoot().setLocale(new Locale(language));
}
```

9.Securitatea

Aplicația implementează un sistem de control al accesului pe bază de roluri. Utilizatorul completează un formular de autentificare care aplică un hash pe parolă și verifică perechea utilizator-parolă cu intrările din baza de date. Dacă utilizatorul este identificat, un obiect "User" este preluat din baza de date, acesta conținând mai multe informații despre utilizator.

UserLoginView.java

```
public void login()
{
    DataAccessAPI api=new DataAccessAPI();
    User user=null;
    if(!Validator.stringContains(username,""))
        user=api.getUser(username, password.hashCode());
}
```

DataAccessAPI.java

```
public User getUser(String username,int password)
{
    User user=new User();
    PreparedStatement prepStmt
    = Database.getPreparedStatement("select id,year,role from users where username=? and password=?");
    ResultSet rs;
    try {
        prepStmt.setString(1, username);
        prepStmt.setInt(2, password);
        rs = prepStmt.executeQuery();

        if(rs.next())
        {
            user.setUsername(username);
            user.setFirstName();
            user.setId(rs.getInt(1));
            user.setYear(rs.getString(2));
            user.setType(rs.getString(3));

            return user;
        }
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return null;
}
```

Dacă autentificarea este cu succes un obiect "SessionObject" este creat și este asociat sesiunii. Acesta conține o instanță a obiectului "User" și este folosit pentru viitoare referiri.

```
if(user!=null)
{
    SessionObject session=new SessionObject();
    session.setUser(user);
    ((HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest()).getSession().setAttribute("user",user);
    ((HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest()).getSession().setAttribute("sessionObject",session);
}
```

Accesul la pagină se face pe bază de roluri. La randarea paginii este apelată o funcție care face verificarea rolurilor utilizatorului.

XHTML

```
<f:metadata>
  <f:event type="preRenderView" listener="#{security.isAuthenticated()}" />
</f:metadata>
```

Security.java

```
public static boolean isAuthenticated()
{
    if(!(checkRole("Student")||checkRole("Profesor")))
    {
        //System.out.println("REDIRECTING");
        redirect("login.xhtml");
        return false;
    }
    return true;
}
```

Accesul în aplicație este de mai multe tipuri și anume:

- Student (doar studenții pot accesa resursa respectivă)
- Profesor (rol folosit pentru paginile profesorilor și butoane adiționale)
- Autentificat (atât studenții cât și profesorii au acces la pagină)
- Proprietar (folosit pentru restricționarea accesului la nivel de proiect, doar proprietarul proiectului poate să își suprascrie proiectul)

SessionObject.java

Din dorința de a reduce numărul de entități asociate direct la sesiune a fost creată clasa "SessionObject" care conține mai multe entități și metode necesare la nivel de sesiune și care simplifică modul în care obiectele asociate sesiunii sunt preluate. Instanța clasei "SessionObject" este accesată cu ajutorul metodei "getSession()" din clasa "Security".

Deoarece resursele aplicației sunt confidențiale ele nu pot fi accesate direct din URL. Accesul la resurse se face progresiv iar clasa SessionObject ține evidența resurselor accesate.

Security.java

Clasa "Security" este o clasa statică responsabilă cu verificarea rolurilor, redirectari și managementul sesiunii.

XHTML

```
<h:form>
  <p:commandLink value="Logout" actionListener="#{security.logout}" />
</h:form>
```

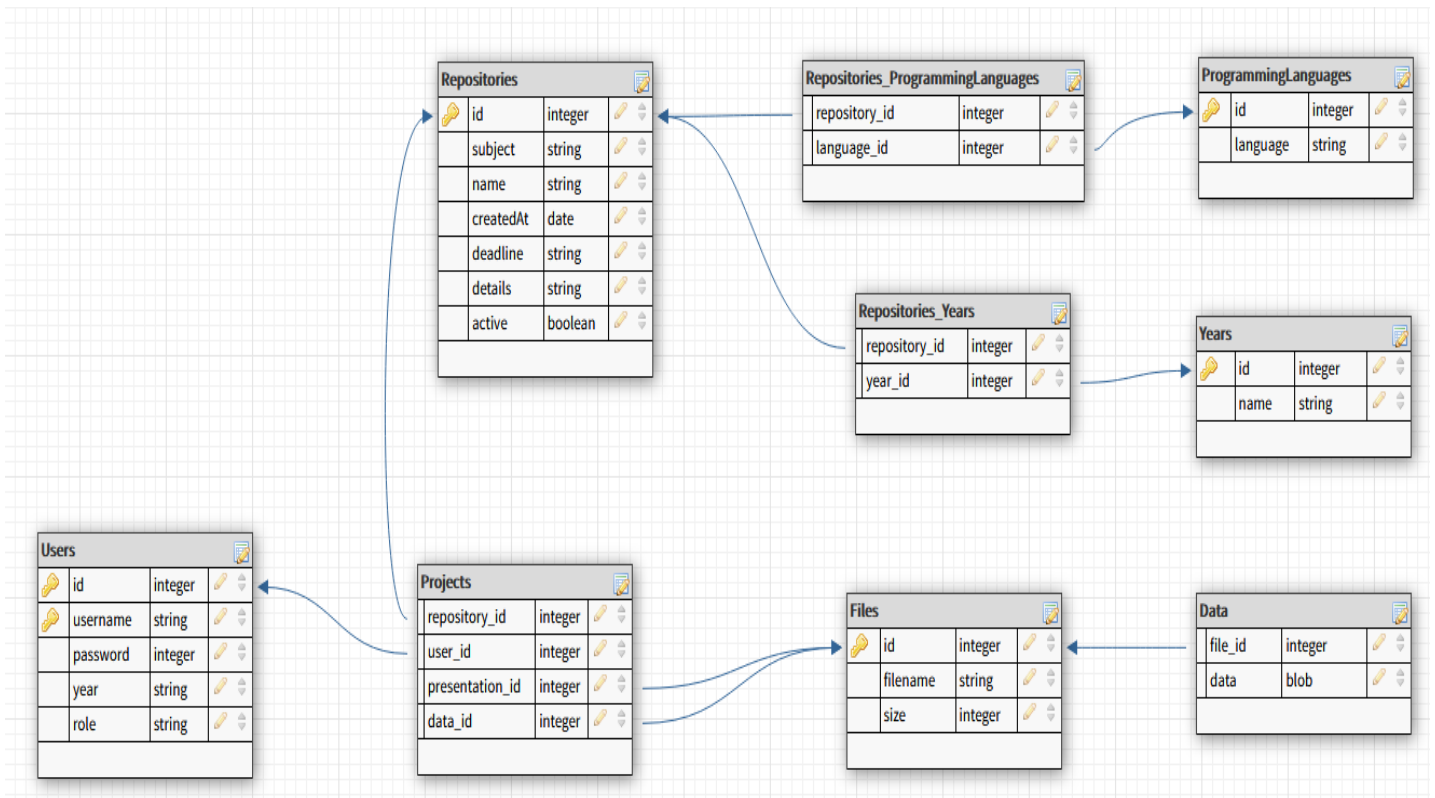
Security.java

```
public static void authorize(String role)
{
    System.out.println("authorize "+role);
    if(!checkRole(role))
        redirect("login.xhtml");
}

public static void logout()
{
    System.out.println("Logged out!");
    ((HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest()).getSession().invalidate();
    redirect("login.xhtml");
}
```


10.Baza de date

Schema bazei de date:



Ca SGBD proiectul folosește PostgreSQL.

Initializarea bazei de date se face prin rularea fișierului “PostgreSQL config.sql” care șterge toate tabelele cu a căror nume coincide cu cele care urmează să fie create.

Accesul la baza de date:

Accesul la baza de date se face prin 3 clase și anume "Database", "DataAccessAPI" și "ViewDataAccess". Clasa Database este o clasa statică la nivel de aplicație, ea oferă funcționalități de baza precum crearea de conexiuni, execuția de comenzi de citire și de scriere.

Clasa DataAccessAPI este orientată în mod special înspre operațiunile de Write, Update și Delete și lucrează pe entitățile de baza asemănătoare cu cele din baza de date.

```
public class DataAccessAPI {  
    public DataAccessAPI()  
    public List<An> getYears()  
    public List<Limbaaj> getLanguages()  
    public List<Limbaaj> getLanguagesOfRepository(int repositoryId)  
    public List<An> getYearsOfRepository(int repositoryId)  
    public boolean addRepository(Repository repository)  
  
    public Repository getRepositoryById(int id)  
    public boolean updateRespositoryActive(int id,boolean value)  
    public User getUser(String username,int password)  
    public List<RepositoryCardView> getRepositoriesByStatusAndYear(boolean status,String year)  
    public List<RepositoryCardView> getRepositoriesByStatus(boolean status)  
    private int addFile(_File file)  
    public boolean addProject(Project project)  
    public boolean deleteFile(int id)  
    public boolean deleteProject(int repositoryId,int userId)  
}
```

Clasa ViewDataAccess este orientată mai mult înspre operațiunile de read parțial, construcția de obiecte asociate direct la frontend.

```
public class ViewDataAccess {  
  
    public ViewDataAccess()  
  
    public List<ProjectDTO> getProjectsOfRepository(int repositoryId)  
  
    public _File getFileById(int id)  
  
    public ProjectView getProject(int repositoryId,int userId)  
  
    public boolean projectExists(int repositoryId,int userId)  
  
}
```

Operatiile asupra bazei de date se fac prin execuție de comenzi cu parametrii.

```
try {  
    prepStmt.setInt(1, project.getRepository().getId());  
    prepStmt.setInt(2, project.getUser().getId());  
    prepStmt.setString(3, project.getDescription());  
    if(presentationId!=-1)  
        prepStmt.setInt(4,presentationId);  
    else  
        prepStmt.setNull(4,Types.INTEGER);  
    if(dataId!=-1)  
        prepStmt.setInt(5,dataId);  
    else  
        prepStmt.setNull(5,Types.INTEGER);  
  
    if(preparedStatement.executeUpdate()>0)  
    {  
        preparedStatement.close();  
        return true;  
    }  
  
    preparedStatement.close();  
  
} catch (SQLException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}  
return false;
```

Ambele clase "DataAccessAPI" și "ViewDataAccess" utilizează clasa statică "Database".

11. Validarea datelor

Validarea datelor se face atât pe frontend cât și pe backend. Validarea de pe frontend se face prin specificarea constrângerilor de lungime dar și prin Regex.

```
<p:inputText id="numeRepository" value="#{repositoryDTO.numeRepository}" size="24" required="true" maxlength="30">
    <f:validateRegex pattern="((?!')(?!\\\/)(?!\\)(?!:)(?!\\*)(?!\\?)(?!\\&#34;)(?!&#60;)(?!&#62;)(?!\\|).)*$"></f:validateRegex>
</p:inputText>
```

Clasele din pachetul "View" care aparțin formularelor au o metodă "objectIsValid" care verifică fiecare proprietate individual și întoarce "true" dacă obiectul este valid. Dacă obiectul este invalid metodă adaugă în răspuns un mesaj de eroare corespunzător și întoarce "false".

```
private boolean objectIsValid()
{
    boolean valid=true;
    ArrayList<FacesMessage> messages=new ArrayList<FacesMessage>();

    if(materie==null)
    {
        valid=false;
        messages.add(new FacesMessage(FacesMessage.SEVERITY_ERROR, "Validation Failed \n \"Materie\" is null",""));
    }

    if(materie.length()>30)
    {
        valid=false;
        messages.add(new FacesMessage(FacesMessage.SEVERITY_ERROR, "Validation Failed \n \"Materie\" is too long",""));
    }

    if(Validator.stringContains(materie,"","/", "\\",";", "*", "?", "\\", "<", ">", "|"))
    {
        valid=false;
        messages.add(new FacesMessage(FacesMessage.SEVERITY_ERROR, "Validation Failed \n \"Materie\" containing unallowed characters",""));
    }

    if(numeRepository==null)
    {
        valid=false;
        messages.add(new FacesMessage(FacesMessage.SEVERITY_ERROR, "Validation Failed \n \"Nume Repository\" is null",""));
    }

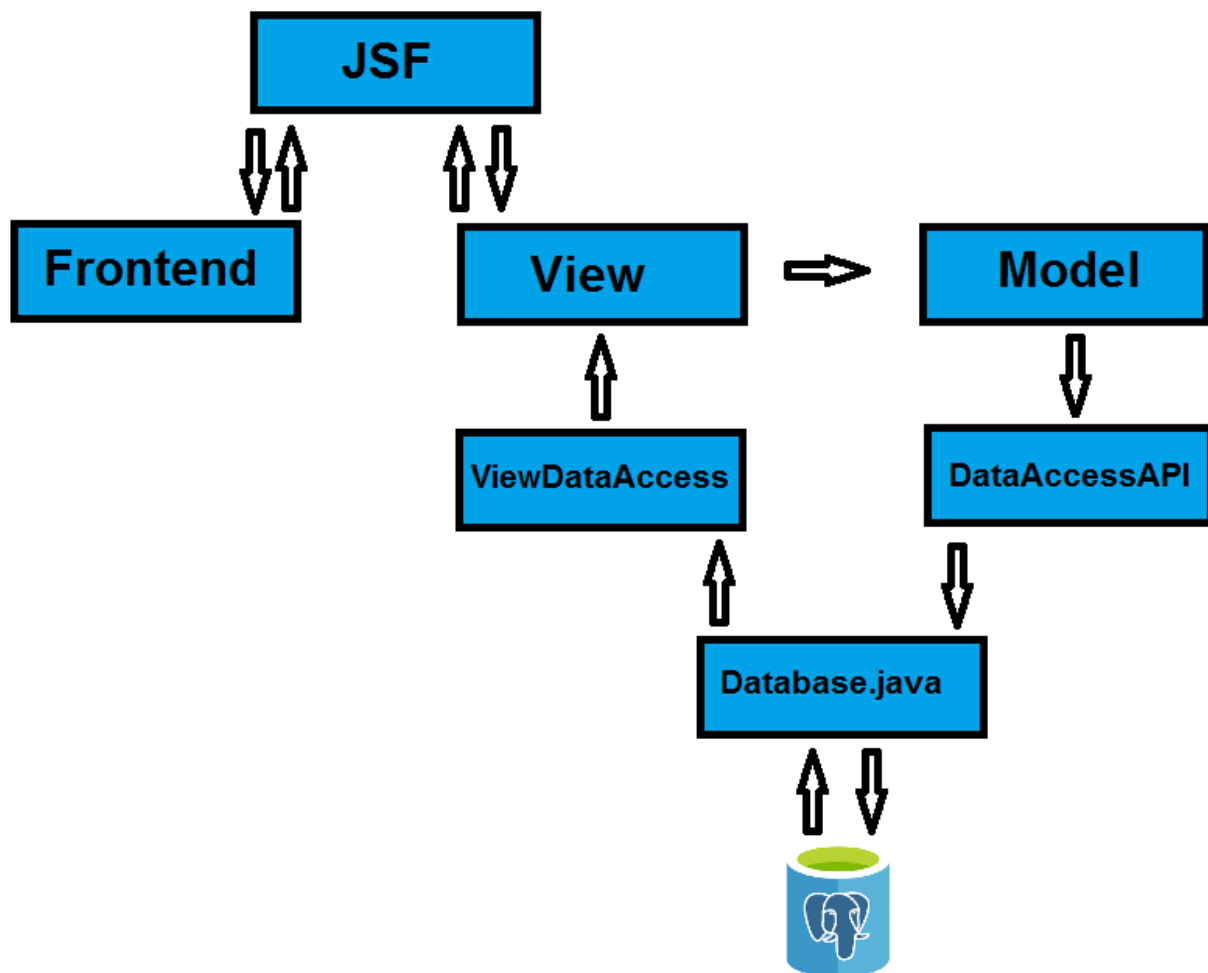
    if(Validator.stringContains(numeRepository,"","/", "\\",";", "*", "?", "\\", "<", ">", "|"))
    {
        valid=false;
        messages.add(new FacesMessage(FacesMessage.SEVERITY_ERROR, "Validation Failed \n \"Nume Repository\" containing unallowed characters",""));
    }

    if(numeRepository.length()>30)
    {
        valid=false;
        messages.add(new FacesMessage(FacesMessage.SEVERITY_ERROR, "Validation Failed \n \"Nume Repository\" is too long",""));
    }
}
```

Verificarea caracterelor “de scapare” se face prin metoda “stringContains” din clasa “Validator”.

12.Preluarea și afișarea datelor

Schema date



La accesarea paginii, clasele din view sunt instantiate iar datele necesare sunt preluate din baza de date prin intermediul clasei "ViewDataAccess".

După completarea formularelor datele sunt validate și apoi transpuse în obiecte din pachetul "Model" prin intermediul metodei "getAsObject()". Datele sunt apoi stocate prin intermediul clasei "DataAccessAPI".

```
private Repository getAsProject()
{
    Repository project=new Repository();
    project.setActiv(true);
    project.setAni(aniSelectati);
    project.setData(getDeadlineDate());
    project.setDetalii(detalii);
    project.setLimbaje(limbajeSelectate);
    project.setMaterie(materie);
    project.setNumeRepository(numerepository);

    project.setCreatedAt(new Date());

    return project;
}
```

13.Manipularea fișierelor de dimensiuni mari

Preluarea fișierelor de la utilizator se face sub formă de "multipart/form-data". Acestea sunt asociate unui obiect de tip "Part" care conține mai multe informații precum numele fișierului, dimensiunea și "input stream".

```
<h:form enctype="multipart/form-data" id="panel">
  <p:growl id="growl" life="10000" />

  <h:panelGrid columns="1" width="100%" cellpadding="10px">

    <h:panelGroup>
      <div class="white-roundEdges-div">
        <dl>
          <dt>Prezentare</dt>

          <dd>
            <h:inputFile value="#{projectUploadView.presentation}">
            </h:inputFile>
          </dd>
        </dl>
      </div>
    </h:panelGroup>
    <h:panelGroup>
      <div class="white-roundEdges-div">
        <dl>
          <dt>Proiect</dt>

          <dd>
            <h:inputFile value="#{projectUploadView.file}" />
          </dd>
        </dl>
      </div>
    </h:panelGroup>
    <h:commandButton class="btn btn-primary" value="#{msg['submit']}"
      icon="ui-icon-check" type="submit"
      action="#{projectUploadView.upload}" update="panel" />

  </h:panelGrid>
</h:form>
```

Fișierul este apoi asociat unui obiect de tip file care este stocat în baza de date respectiv în tabela "files" unde se află numele și dimensiunea fișierului și tabela Data unde șirul de octeți este salvat sub formă de blob.

```
private int addFile(_File file)
{
    PreparedStatement prepStmt= Database.getPreparedStatement("insert into files(filename,size) values(?,?) returning id");
    ResultSet rs;
    int id=-1;

    try {
        prepStmt.setString(1, file.getFilename());
        prepStmt.setLong(2, file.getSize());
        rs=prepStmt.executeQuery();
        if(rs.next())
        {
            id=rs.getInt(1);
        }
        else
            throw new SQLException("Inserting file failed.");
        rs.close();
        prepStmt.close();

        prepStmt= Database.getPreparedStatement("insert into data(file_id,data) values(?,?)");

        if(id!=-1)
            throw new SQLException("Invalid id.");
        prepStmt.setInt(1, id);
        prepStmt.setBinaryStream(2, file.getFs(),file.getSize());

        if(prepStmt.executeUpdate()>0)
        {
            prepStmt.close();
            file.getFs().close();

            return id;
        }
        else
            throw new SQLException("Saving byte object failed");
    }
```


Deoarece fișierele de dimensiuni mari îngreunează initializarea claselor din view și ocupă multă memorie se preia din baza de date doar id-ul fișierului, urmând că acesta să fie citit doar la apăsarea butonului de descărcare.

```
public class ProjectView {

    private String userName;
    private String description;
    private int presentationId;
    private int dataId;
    private _File presentation;
    private _File data;

    public StreamedContent getFile() {
        this.data=new ViewDataAccess().getFileById(dataId);
        file=new DefaultStreamedContent(data.getFs(), "application/zip", data.getFilename());
        return file;
    }
}
```

Pentru afișarea prezentării PDF în pagină se procedează în mod similar cu fișierul însă headerul este modificat corespunzător pentru afișarea în pagină.

```
public void downloadPDF() throws IOException {

    this.presentation=new ViewDataAccess().getFileById(presentationId);

    FacesContext facesContext = FacesContext.getCurrentInstance();

    HttpServletResponse response = (HttpServletResponse) facesContext.getExternalContext().getResponse();
    response.reset();
    response.setContentType("application/pdf");
    response.setContentLengthLong(presentation.getSize());
    response.setHeader("Content-Disposition", "inline; filename=\"" + presentation.getFilename() + "\"");
    OutputStream responseOutputStream = response.getOutputStream();

    InputStream pdfInputStream = presentation.getFs();

    byte[] bytesBuffer = new byte[2048];
    int bytesRead;
    while ((bytesRead = pdfInputStream.read(bytesBuffer)) > 0) {
        responseOutputStream.write(bytesBuffer, 0, bytesRead);
    }
    responseOutputStream.flush();
    pdfInputStream.close();
    responseOutputStream.close();
    facesContext.responseComplete();
}
```

14. Antiplagiere

O analiză la un nivel mai amănunțit ar putea însemna detectarea variabilelor și contextul în care sunt modificate, trasarea unor "hărți" în care variabilele sunt menționate și folosite.

Detectarea similarității între elemente presupune detectarea similarității între metode și proprietăți. Astfel redenumirea variabilelor și interschimbarea funcțiilor nu va afecta gradul de similaritate decât într-o oarecare măsură, accentul fiind pus pe tipurile de date și numărul lor.

Detectarea similarității constă în compararea listelor de elemente între ele, element cu element încercând să detecteze elementele cu grad de similaritate cel mai ridicat.

Rezultatul parsării este o lista de obiecte menite să reprezinte clasele, metodele și proprietățile acestora, tipurile de date și parametrii acceptați, inclusiv numele variabilelor și definiția funcțiilor.

Proiectul presupune identificarea tuturor fișierelor din proiect care conțin cod sursă și parsarea lor.

Un proiect a fost început în această direcție momentan disponibil doar pe fișierele sursă aparținând limbajului java.

O oportunitate pe care acest proiect o oferă este integrarea cu un serviciu de detectare al similarităților în fișierele sursă.

Că și parte practică a fost implementat un program care să parseze codul Java și să genereze o astfel de lista.

Pe o clasă de probă a fost obținut rezultatul următor:

```
public class TestClass {  
  
    TestClass()  
    {  
  
    }  
  
    private double x;  
  
    void testMethod(int parametru)  
    {  
        int x;  
        String test="asdsdasd";  
        String test2="";  
        //this is a comment  
  
        /*this is  
        * a  
        * block comment  
        //a comment in a block comment  
        * */  
  
        // /*a commented block comment*/  
  
        String test3="a string that contains //a comment line and /*a block comment*/";  
  
    }  
}
```

Access modifier: public

Class name: TestClass

Members:

Access modifier: private

Type: double

Identifier: x

Methods:

Access modifier: private

Return type: void

Method name: testMethod





Parameters:

Access modifier: default

Type: int

Identifier: parametru

Parsarea codului se face prin definirea unor tokeni.

- ▷  Comment.java
- ▷  Identifier.java
- ▷  Keyword.java
- ▷  Parentheses.java

Fiecare clasă este o instanță a clasei Token, însă toate implementează metoda "match" într-un mod diferit.

```
public abstract class Token {  
    public String type;  
  
    public TokenResult result;  
  
    public abstract boolean match(char[] code,int index);  
}
```

Metoda "match" primește ca argument o listă de caractere reprezentând codul sursă și un index curent. S-a folosit această implementare deoarece este mai eficientă decât crearea unui nou obiect de tip "String" pentru fiecare încercare de potrivire. Că și răspuns metodă întoarce "false" dacă șirul de caractere nu reprezintă tokenul respectiv. Dacă șirul de caractere îndeplinește condiția tokenului, funcția salvează rezultatul în proprietatea "result" și returnează "true".

```

@Override
public boolean match(char[] code, int index) {

    String block="";
    int count=0;
    if(code[index]==parenthesesBegin)
        for(int i=index;i<code.length;i++)
        {
            block+=code[i];
            if(code[i]==parenthesesBegin)
                count++;
            if(code[i]==parenthesesEnd)
                count--;
            if(count==0)
            {
                this.result=new TokenResult(index,index+block.length(),this.type,block);
                return true;
            }
        }
    return false;
}

```

Clasa TokenResult conține indexul de început, indexul de final, tipul tokenului și codul respective care reprezintă tokenul.

```

public class TokenResult {

    public int begin;
    public int end;
    public String type;
    public String value;

    public TokenResult()
    {

    }

    public TokenResult(int begin,int end,String type,String value)
    {
        this.begin=begin;
        this.end=end;
        this.type=type;
        this.value=value;
    }

}

```

Parserul vine cu o funcție de configurare a obiectelor "Token".

```
private void configTokenizer()
{
    Comment token=new Comment();
    token.type="string";
    token.beginSequence="\\";
    token.escapeSequences.add("\n");
    token.escapeSequences.add("\\"");
    token.exception="\\\\";
    tokens.add(token);

    token=new Comment();
    token.type="comment";
    token.beginSequence="//";
    token.escapeSequences.add("\n");
    tokens.add(token);

    token=new Comment();
    token.type="block comment";
    token.beginSequence="/*";
    token.escapeSequences.add("*/");
    tokens.add(token);

    Keyword keyword=new Keyword();
    keyword.type="access modifier";
    keyword.keywords.add("public");
    keyword.keywords.add("protected");
    keyword.keywords.add("private");
    tokens.add(keyword);
}
```

Parsarea codului rezultă într-o lista de obiecte de tip "TokenResult". Această lista este apoi parcursă într-un mod similar încercând să se aplice reguli de sintaxa.

Sintaxa este alcătuită din obiecte de tip "SyntaxRule". Această clasă conține tipul detectat, un obiect de tip "ParsedResult" care reprezintă obiectul parsat, o listă de obiecte de tip "Syntax" și o funcție "match".

```
public abstract class SyntaxRule {

    public String type;
    public ParsedResult result;
    public int tokensLength;
    public List<Syntax>syntaxes=new LinkedList<Syntax>();

    public abstract boolean match(List<TokenResult> tokens,int index);

}
```

În lista de sintaxe se află liste de tipuri de tokeni așezați în ordine pentru a alcătui obiectul vizat. Dacă una din sintaxe se potrivește se construiește un obiect de tip "ParsedResult" iar valoarea "true" este returnată.

Asemănător funcției de configurare a tokenilor, este necesară o funcție de configurare a sintaxei.

```
private void configParser()
{
    SyntaxRule rule=new ClassSyntax();

    Syntax syntax=new Syntax();
    syntax.syntax.add("access modifier");
    syntax.syntax.add("class");
    syntax.syntax.add("identifier");
    syntax.syntax.add("block");
    rule.syntaxes.add(syntax);

    syntax=new Syntax();
    syntax.syntax.add("class");
    syntax.syntax.add("identifier");
    syntax.syntax.add("block");
    rule.syntaxes.add(syntax);

    rules.add(rule);
}
```

Rezultatul final al parsarii este lista de obiecte de tip "ParsedResult", obiecte care urmează să fie comparate.

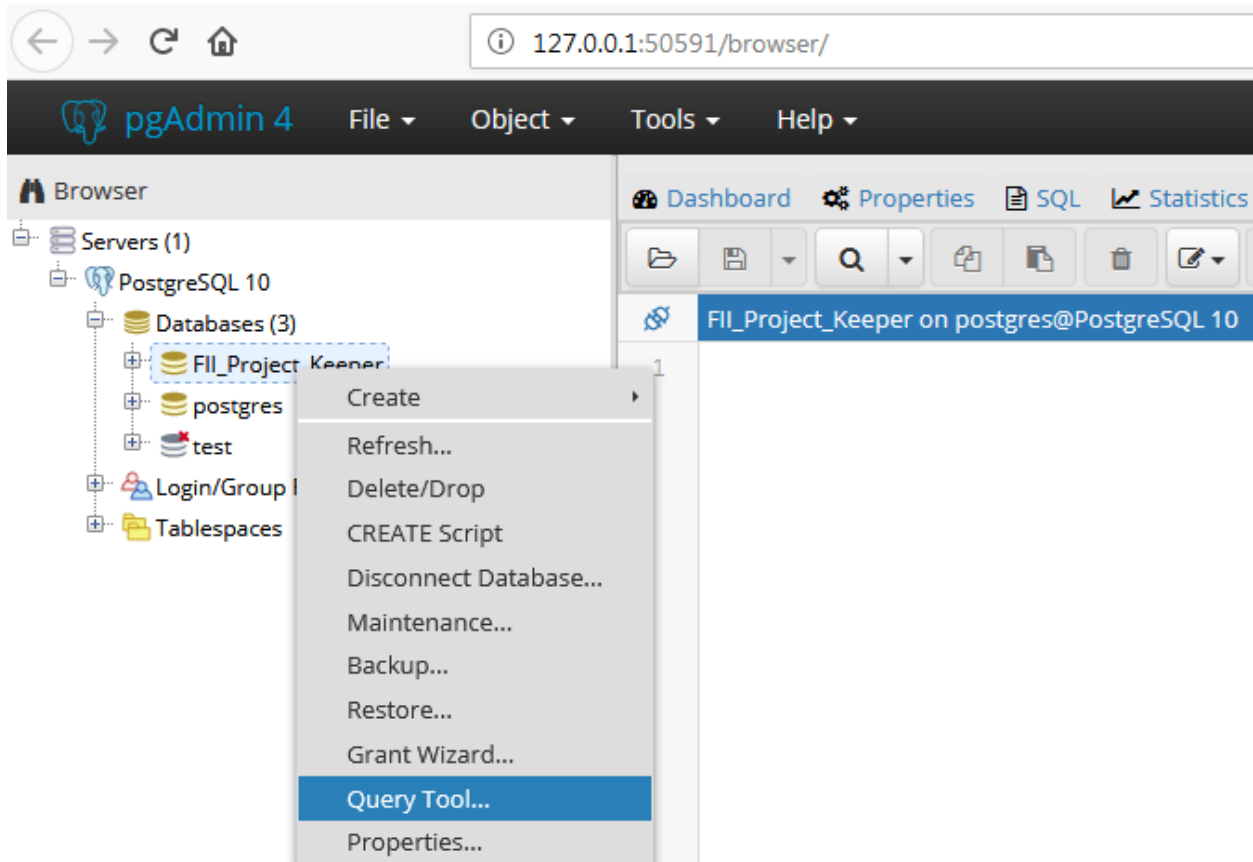
15. Manual de utilizare

Ghid pentru instalare:

Pentru instalarea aplicației este nevoie de o serie de programe și anume:

- Java SE Runtime Environment 8
- O bază de date în PostgreSQL
- O instanță a unui server GlassFish, preferabil Eclipse GlassFish 5.1
- Un program care să facă *deploy* la aplicație pe server, preferabil Eclipse EE

Baza de date se configurează prin rularea scriptului “PostgreSQL config.sql”. Comenzile se pot rula din “Query Tool” oferit de interfață web din programul pgAdmin.



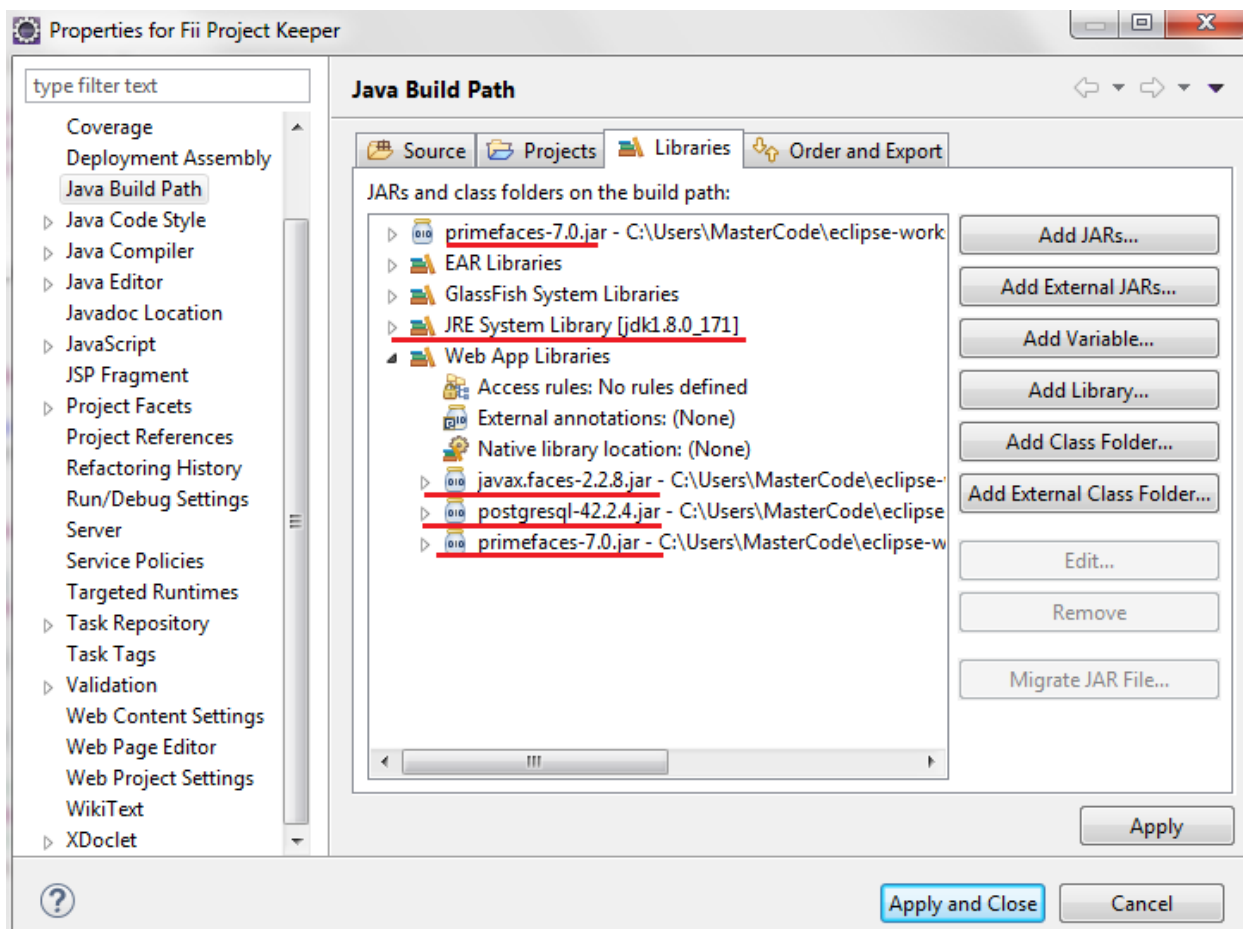
Este necesară modificarea clasei “Database.java” conform configurației bazei de date instalate.

În mod implicit clasa folosește următoarea configurație:

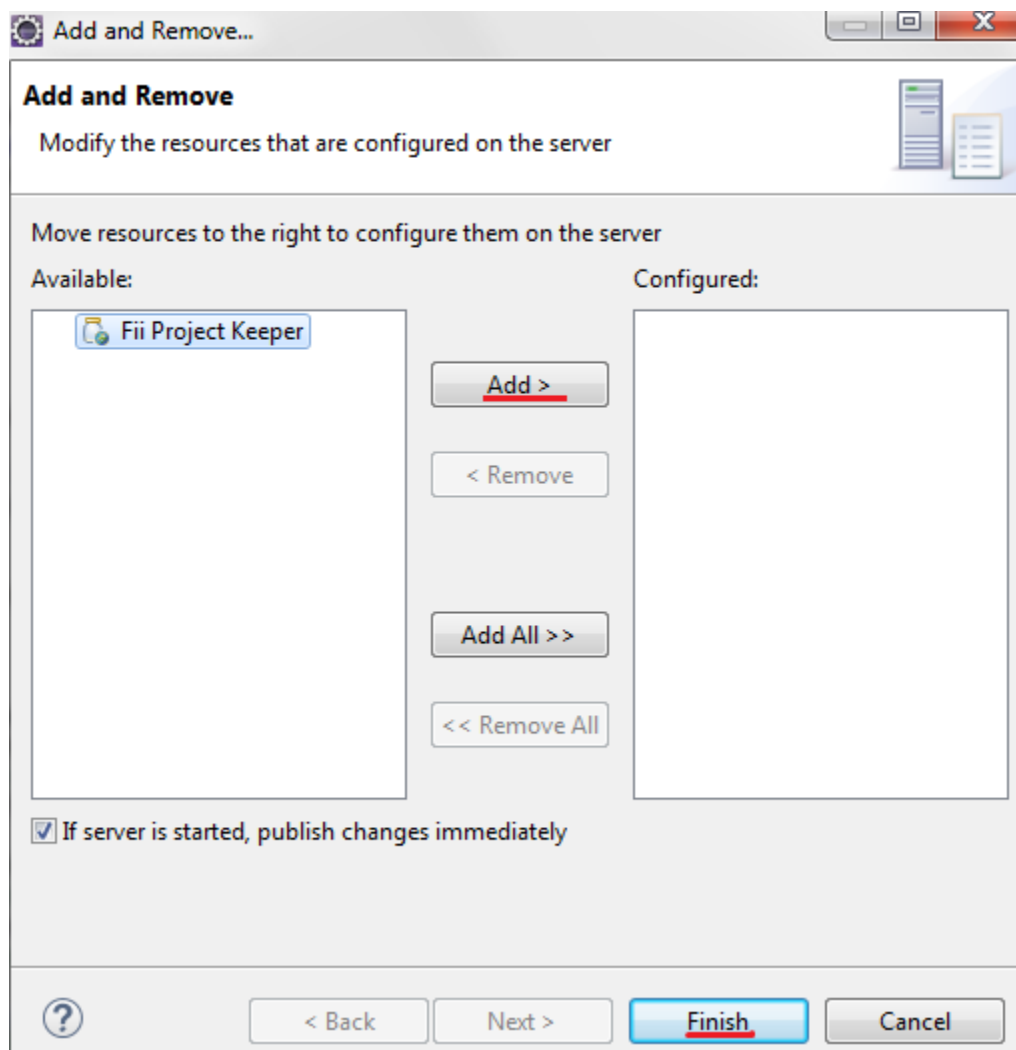
```
"jdbc:postgresql://127.0.0.1:5432/FII_Project_Keeper", "postgres", "sql"
```

Popularea bazei de date se face prin execuția metodei main din clasa “ConfigureDatabase.java”.

Este necesară reconfigurarea căilor din *Build Path* conform programelor instalate în sistem.

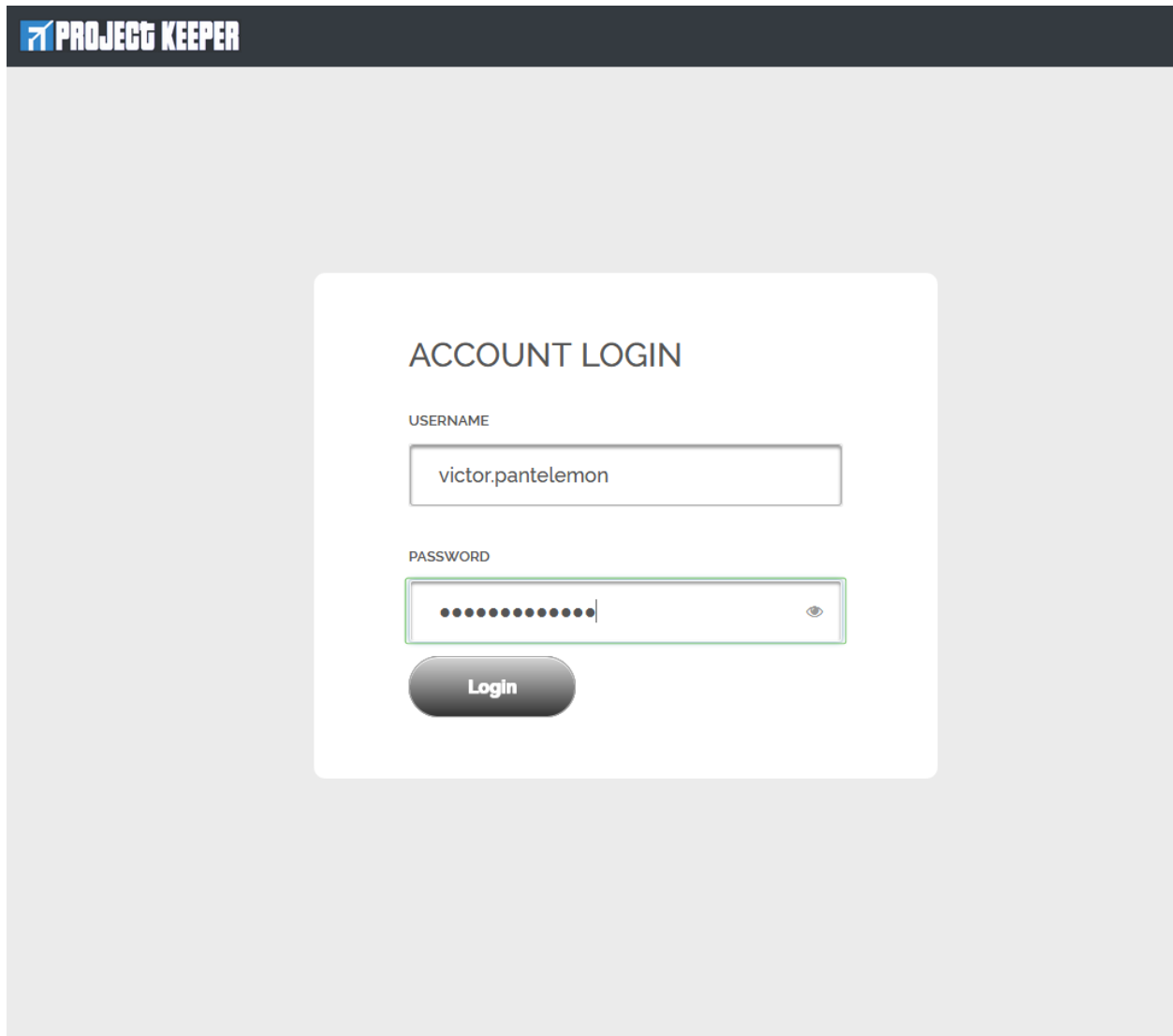


Ca pas final este adaugarea aplicației pe server și pornirea acestuia.



Ghid de utilizare:

Autentificarea în aplicație se face pe pagina “login.xhtml”.



The screenshot shows the 'ACCOUNT LOGIN' interface for 'PROJECT KEEPER'. The page has a dark header with the logo. The login form is centered on a light gray background. It includes a 'USERNAME' field with the text 'victor.pantelemon' and a 'PASSWORD' field with masked characters and a toggle icon. A 'Login' button is positioned below the password field.

PROJECT KEEPER

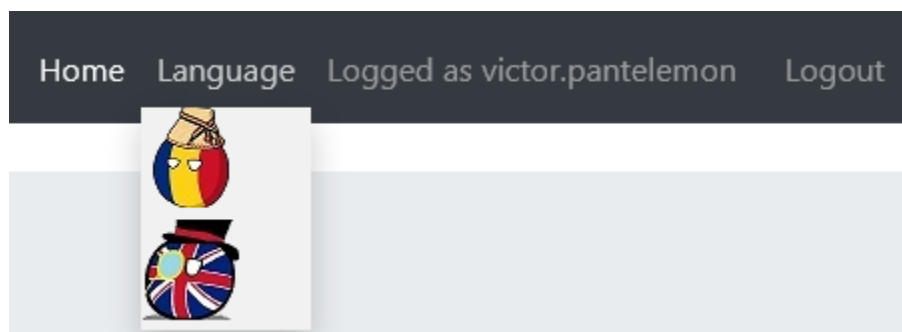
ACCOUNT LOGIN

USERNAME

PASSWORD

Login

Limba în aplicație se selectează din bara de navigație din secțiunea “Language”.



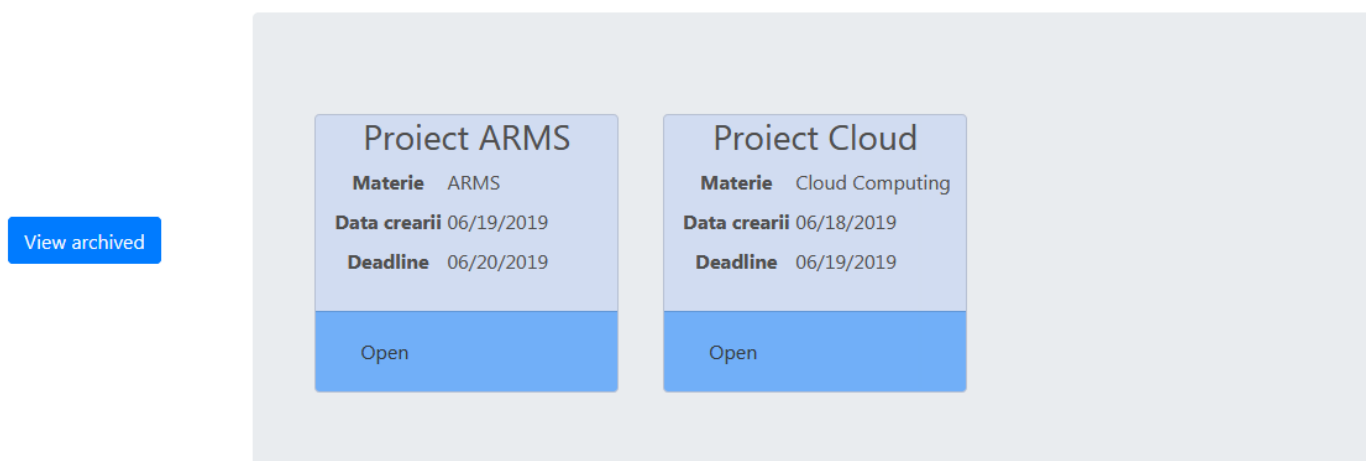
Eliminarea sesiunii se face prin apăsarea butonului “Logout”.

Utilizatorul în calitate de student poate vizualiza proiectele din care face parte accesând pagina “repositories.xhtml”

Bine ai venit, Victor!

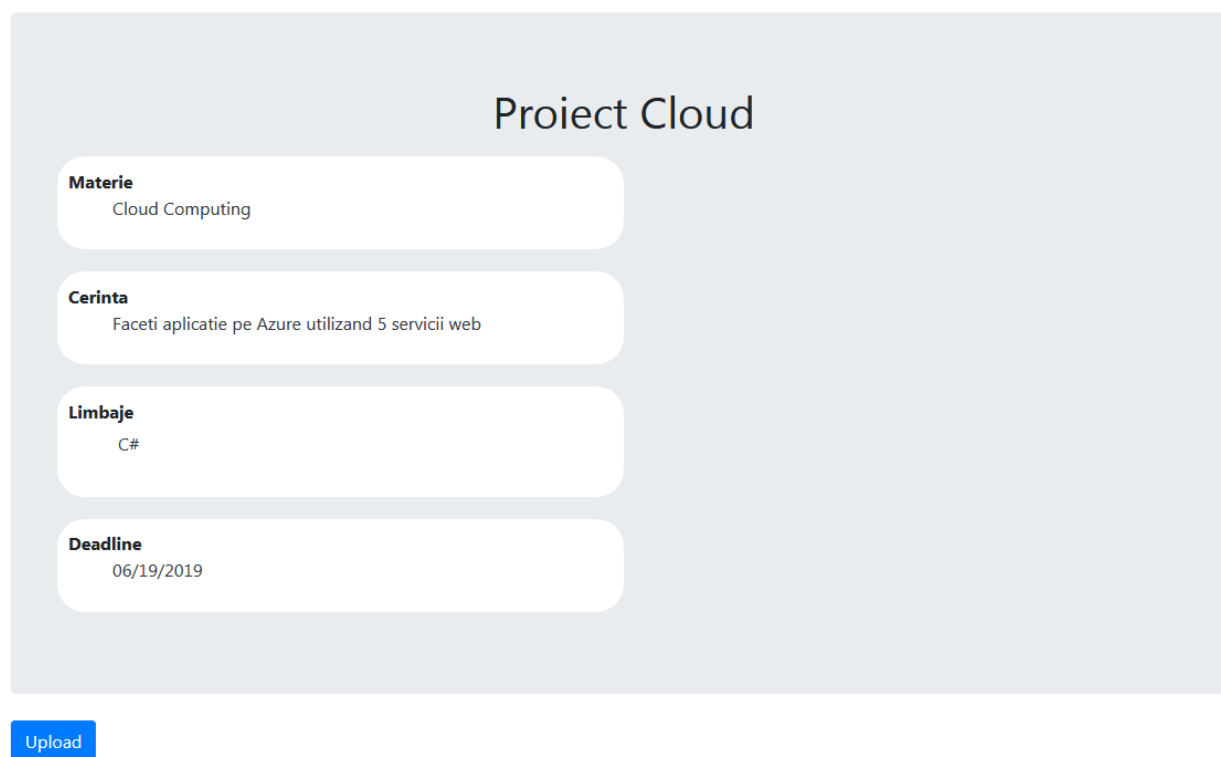


Aici utilizatorul vede temele de proiect active din care face parte. Dacă nici o temă nu este prezentă se va arată un mesaj corespunzător.



Butonul “View archived” afișează temele de proiect închise din care studentul a făcut parte.

Prin apăsarea butonului “Open” utilizatorul este redirecționat pe o pagină unde se pot vedea mai multe detalii despre tema de proiect propusă.



Utilizatorul apoi completează un formular care trebuie să conțină o descriere a propriului sau proiect, opțional un fișier PDF care să conțină o prezentare a temei și o arhivă zip conținând codul sursă al proiectului.

Incarca un proiect

Descriere

Prezentare

No file selected.

Proiect

No file selected.

Utilizatorul își poate vedea propriul proiect, poate descărca fișierele sale și îl poate suprascrie atâta timp cât tema de proiect este încă activă.

Utilizatorul în calitate de profesor poate propune teme de proiect prin completarea formularului următor.

Incarca o tema de proiect

Informatii principale	Nume repository *		<input type="text"/>
Audienta*	Alege anii	<input type="text"/>	
Termen incarcare	Data:	<input type="text"/>	
Alte informatii	Materia	Alege Materia	<input type="text"/>
	Limbaje recomandate	<input type="text"/>	
	Alte informatii	<input type="text"/>	
<input type="button" value="✓ Incarca"/>			

Profesorul poate vedea toate propunerile de proiect care sunt sau au fost active. Și poate închide și redeschide o temă.

Proiect TSP.NET

Materie

TSP.NET

Cerinta

Faceti un class library cu un windows forms si host wcf si client wcf si cu wpf si asp.net.

Limbaje

C#

Deadline

06/20/2019

[View Projects](#)

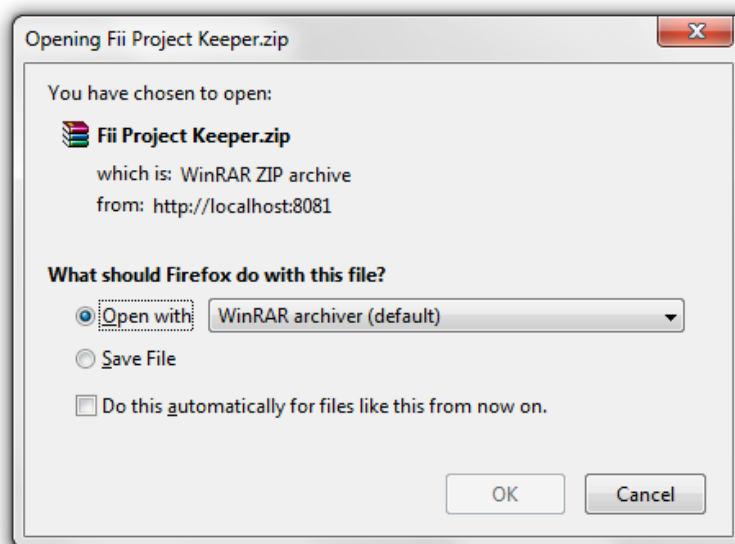
[Restore](#)

Asemenea studentului, profesorul poate încarcă un proiect și în plus are acces la toate proiectele care au fost încărcate.

Utilizator	Descriere proiect
victor.pantelemon	Am dezvoltat o aplicatie pe Azure care face managementul unui restaurant... Open

View Presentation

Download Project



Concluzii

Ca și concluzie, în urma efortului depus a rezultat o aplicație funcțională care îndeplinește scopul propus și anume cel de a oferi un mediu centralizat în care atât studenții cât și profesorii să își stocheze proiectele.

În opinia mea o astfel de aplicație ar fi utilă în facultate. Aplicația deschide multe oportunități și are mult potențial de dezvoltare.

Ca și direcții viitoare aplicația poate veni cu o sumedenie de îmbunătățiri și anume:

- Dezvoltarea unui sistem de detectare al lucrărilor asemănătoare conform specificațiilor menționate în capitolele anterioare.
- Posibilitatea de a stoca și teme de laborator, teme temporare și referate.
- Utilizarea unui serviciu de generare a emailurilor și a notificărilor când apar teme noi.
- Posibilitatea adăugării unui sistem de notare online.
- Dezvoltarea unei pagini de administrare și configurare a aplicației.
- Utilizarea unui ORM.
- Statistici și metrice pe baza proiectelor propuse.

Bibliografie

https://profs.info.uaic.ro/~acf/tj/slides/jsf_slide_en.pdf

https://ro.wikipedia.org/wiki/JavaServer_Faces

<https://en.wikipedia.org/wiki/PrimeFaces>

<https://www.primefaces.org/showcase/>

<https://stackoverflow.com/>

https://www.tutorialspoint.com/jsf/jsf_composite_components.htm

https://app.dbdesigner.net/designer/schema/guest_template

<https://www.youtube.com/watch?v=TifG->

[hPO8IU&list=PLEAQNNR8IIB4S8nNUI50ArfgU1nXlhdRu](https://www.youtube.com/watch?v=TifG-hPO8IU&list=PLEAQNNR8IIB4S8nNUI50ArfgU1nXlhdRu)

<https://www.mkymong.com/jsf2/jsf-2-internationalization-example/>

<https://examples.javacodegeeks.com/enterprise-java/jsf/jsf-internationalization-example/>

<https://stackoverflow.com/questions/6109138/jsf-locale-change-listener-not-persistent>

<https://javaparser.org/>