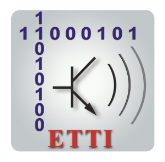
**UNIVERSITATEA POLITEHNICA din BUCUREȘTI**

Facultatea de **Electronică, Telecomunicații și Tehnologia Informației**

***Proiect PAI***

****

-București 2019-

|  |  |
| --- | --- |
|  | *Student:Manea Vlad Cristian* |

Baza de Date:

Baza de date este dezvoltata folosind MySql. Tabelele sunt următoarele:

* Products (idProducts (PK), Code, Date, Price);
* Components (idComponents (PK), Name, Model, Manufacturer, Costs, Parameters);
* Assemblyline (idAssemblyline (PK), idProducts (FK), idComponents (FK), Validation);

Diagrama logica a bazei de date “factory” este prezentata in figura 1.

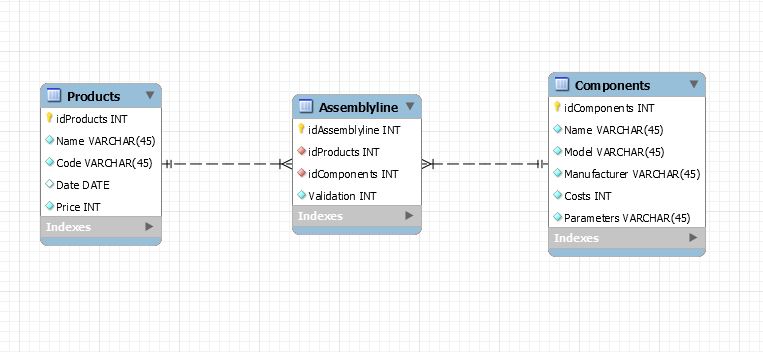


Figura 1 (Diagrama logica)

Intre tabelele Products si Components se realizeaza o relatie M:N prin intermediul tabelei de legatura Assemblyline.

Dezvoltarea unei aplicatii web cu ajutorul tehnologiei Java Server Pages (JSP)

Aplicatia este construita in NetBeans folosind libraria MySQL JDBC Driver si conexiunea este facuta prin intermediul serverului GlasFish Server 4.1.1.

Initial am creat clasa JavaBean si am creat si pachetul “db” care contine toate functiile care vor da functionalitate aplicatiei web: conexiunea la baza de date , deconectarea ,adaugarea de elemente ,modificarea elementelor ,afisarea elementelor deja existente si stergerea acestora.In JavaBean se importa java.sql.\* pentru a putea folosi Clasele Connection ,Statement ,ResultSet si metodele acestora.

JavaBean

package db;

import java.sql.\*;

public class JavaBean {

String error; //In error se vor memora mesajele de eroare.

Connection con; //Prin intermediul lui con se creaza legatura cu baza de date.

public JavaBean() {

}

public void connect() throws ClassNotFoundException, SQLException, Exception {

try {

Class.forName("com.mysql.jdbc.Driver");

con = DriverManager.getConnection("jdbc:mysql://localhost:3306/factory?useSSL=false", "root", "root"); //Se creaza conexiunea cu baza de date.

} catch (ClassNotFoundException cnfe) {

error = "ClassNotFoundException: Nu s-a gasit driverul bazei de date.";

throw new ClassNotFoundException(error);

} catch (SQLException cnfe) {

error = "SQLException: Nu se poate conecta la baza de date.";

throw new SQLException(error);

} catch (Exception e) {

error = "Exception: A aparut o exceptie neprevazuta in timp ce se stabilea legatura la baza de date.";

throw new Exception(error); //Sunt tratate erorile cele mai frecvente.

}

}

public void disconnect() throws SQLException {

try {

if (con != null) {

con.close(); //Deconectarea de la baza de date.

}

} catch (SQLException sqle) {

error = ("SQLException: Nu se poate inchide conexiunea la baza de date.");

throw new SQLException(error);

}

}

Sunt scrise functii de adaugare pentru fiecare tabela.

public void adaugaProducts(String Name, String Code, String Date,int Price)

throws SQLException, Exception {

if (con != null) {

try {

Statement stmt; //Prin intermediul lui stmt se fac adaugari si modificari la baza de date;

stmt = con.createStatement();

stmt.executeUpdate("insert into Products(Name, Code, Date, Price) values('" + Name + "' , '" + Code + "', '" + Date + "', '" + Price + "');");

} catch (SQLException sqle) {

error = "ExceptieSQL: Reactualizare nereusita; este posibil sa existe duplicate.";

throw new SQLException(error);

}

} else {

error = "Exceptie: Conexiunea cu baza de date a fost pierduta.";

throw new Exception(error);

}

}

public ResultSet vedeTabela(String tabel) throws SQLException, Exception {

ResultSet rs = null; //rs retine datele obtinute de la baza de date in urma unor interogari.

try {

String queryString = ("select \* from `Factory`.`" + tabel + "`;");

Statement stmt = con.createStatement(/\*ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY\*/);

rs = stmt.executeQuery(queryString);

} catch (SQLException sqle) {

error = "SQLException: Interogarea nu a fost posibila.";

throw new SQLException(error);

} catch (Exception e) {

error = "A aparut o exceptie in timp ce se extrageau datele.";

throw new Exception(error);

}

return rs;

}

public void stergeDateTabela(String[] primaryKeys, String tabela, String dupaID) throws SQLException, Exception {

if (con != null) {

try {

long aux;

PreparedStatement delete;

delete = con.prepareStatement("DELETE FROM " + tabela + " WHERE " + dupaID + "=?;");

for (int i = 0; i < primaryKeys.length; i++) {

aux = java.lang.Long.parseLong(primaryKeys[i]);

delete.setLong(1, aux);

delete.execute();

}

} catch (SQLException sqle) {

error = "ExceptieSQL: Reactualizare nereusita; este posibil sa existe duplicate.";

throw new SQLException(error);

} catch (Exception e) {

error = "A aparut o exceptie in timp ce erau sterse inregistrarile.";

throw new Exception(error);

}

} else {

error = "Exceptie: Conexiunea cu baza de date a fost pierduta.";

throw new Exception(error);

}

}

public void modificaTabela(String tabela, String IDTabela, int ID, String[] campuri, String[] valori) throws SQLException, Exception {

String update = "update " + tabela + " set ";

String temp = "";

if (con != null) {

try {

for (int i = 0; i < campuri.length; i++) {

if (i != (campuri.length - 1)) {

temp = temp + campuri[i] + "='" + valori[i] + "', ";

} else {

temp = temp + campuri[i] + "='" + valori[i] + "' where " + IDTabela + " = '" + ID + "';";

}

}

update = update + temp;

Statement stmt;

stmt = con.createStatement();

stmt.executeUpdate(update);

} catch (SQLException sqle) {

error = "ExceptieSQL: Reactualizare nereusita; este posibil sa existe duplicate.";

throw new SQLException(error);

}

} else {

error = "Exceptie: Conexiunea cu baza de date a fost pierduta.";

throw new Exception(error);

}

}

public ResultSet intoarceLinie(String tabela, int ID) throws SQLException, Exception {

ResultSet rs = null;

try {

String queryString = ("SELECT \* FROM " + tabela + " where idProducts=" + ID + ";");

Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY);

rs = stmt.executeQuery(queryString); //sql exception

} catch (SQLException sqle) {

error = "SQLException: Interogarea nu a fost posibila.";

throw new SQLException(error);

} catch (Exception e) {

error = "A aparut o exceptie in timp ce se extrageau datele.";

throw new Exception(error);

}

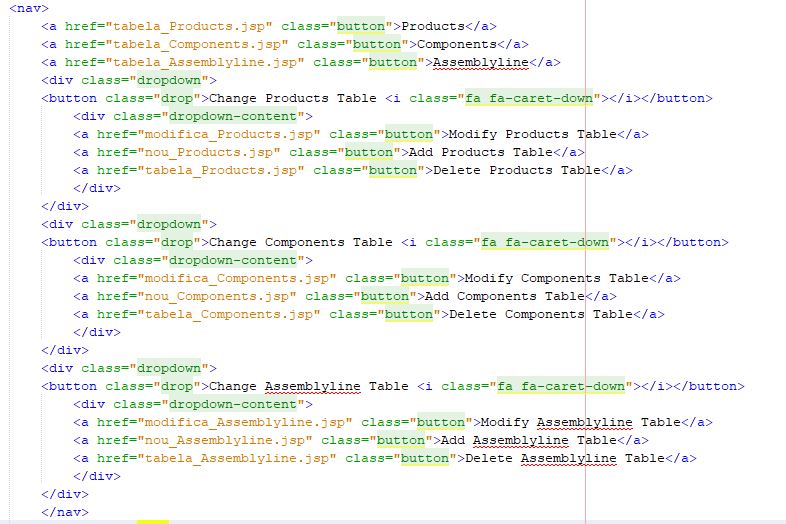
return rs;

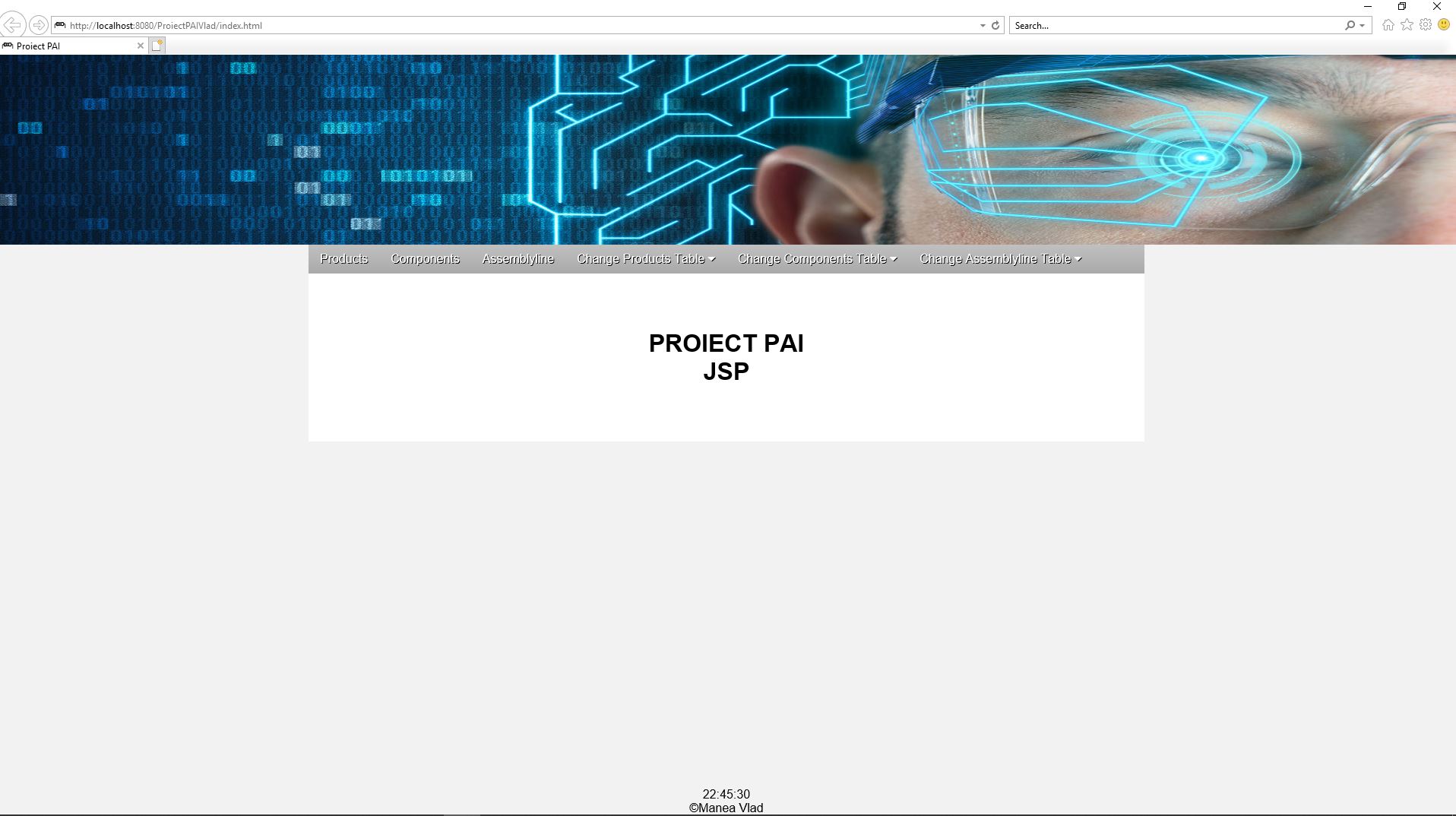
}

}

In cadrul proiectului se genereaza automat pagina “index.html” care este folosita ca o pagina principala din care se pot accesa toate celelalte pagini de tip JSP.

Index.html

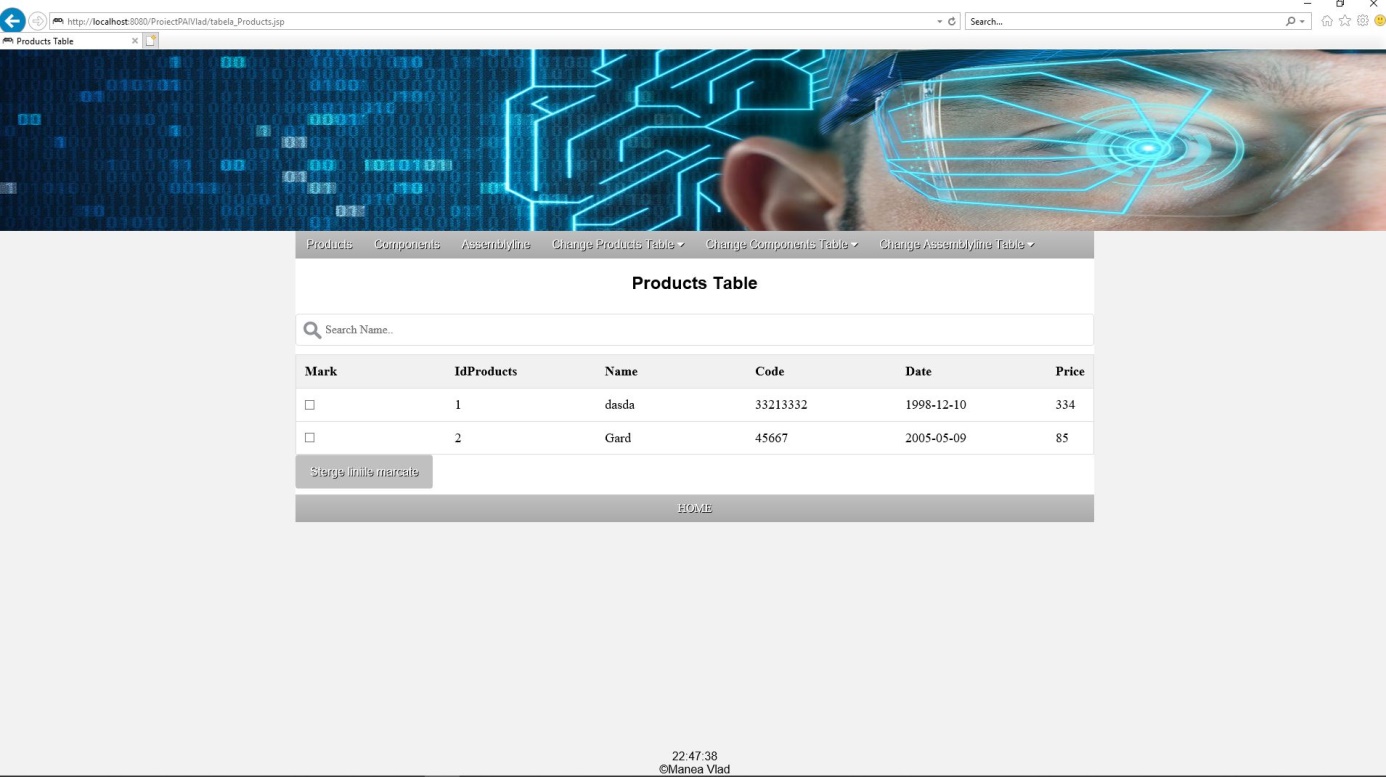




Am creat pagina de tip JSP “tabela\_products.jsp” pentru a vizualiza elementele tabelei Products.In orice pagina in care se folosesc date din baza de date “factory” se stabileste intai o conexiune folosind functia “connect()” ,iar la final se taie aceasta conexiune prin functia “disconnect()”.

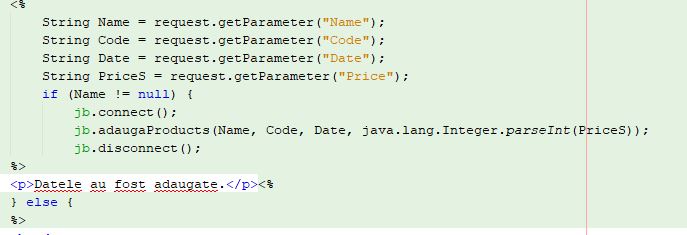
Tabela Products

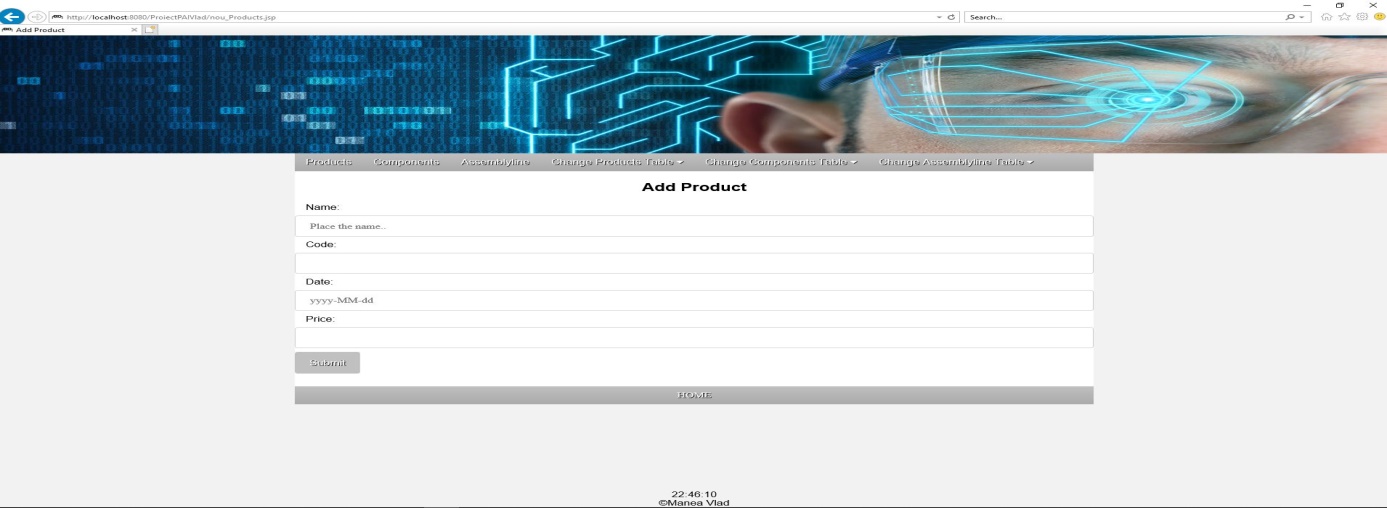




In pagina “nou\_Products.jsp” se adauga date in baza de date prin intermediul input si metodei “getParameter()” ,urmat de utilizarea functiei create in clasa JavaBean “adaugaProducts()”.

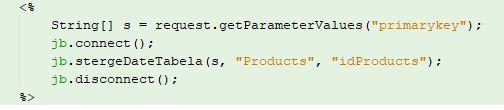
Adauga Product





In pagina “sterge\_Products.jsp” se foloseste functia de stergere definite in clasa JavaBean, iar aceasta este accesata in combinatie cu pagina “tabela\_Products.jsp”.

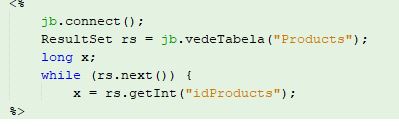
Sterge Product



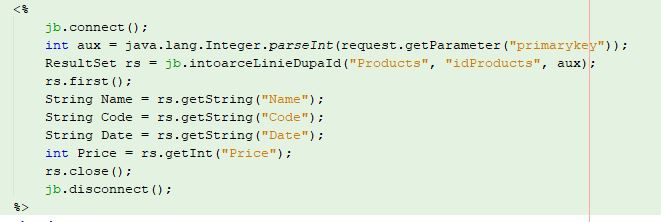


In pagina “modifica\_Products.jsp” este afisata tabela Products si se poate alege linia la care dorim sa facem modificari.

Modifica Product

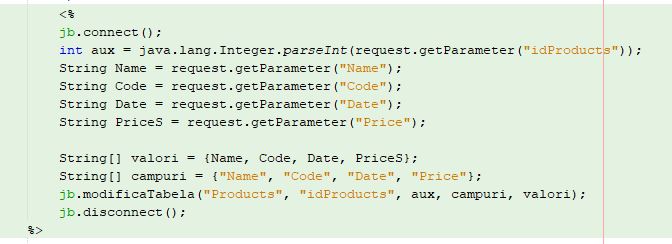


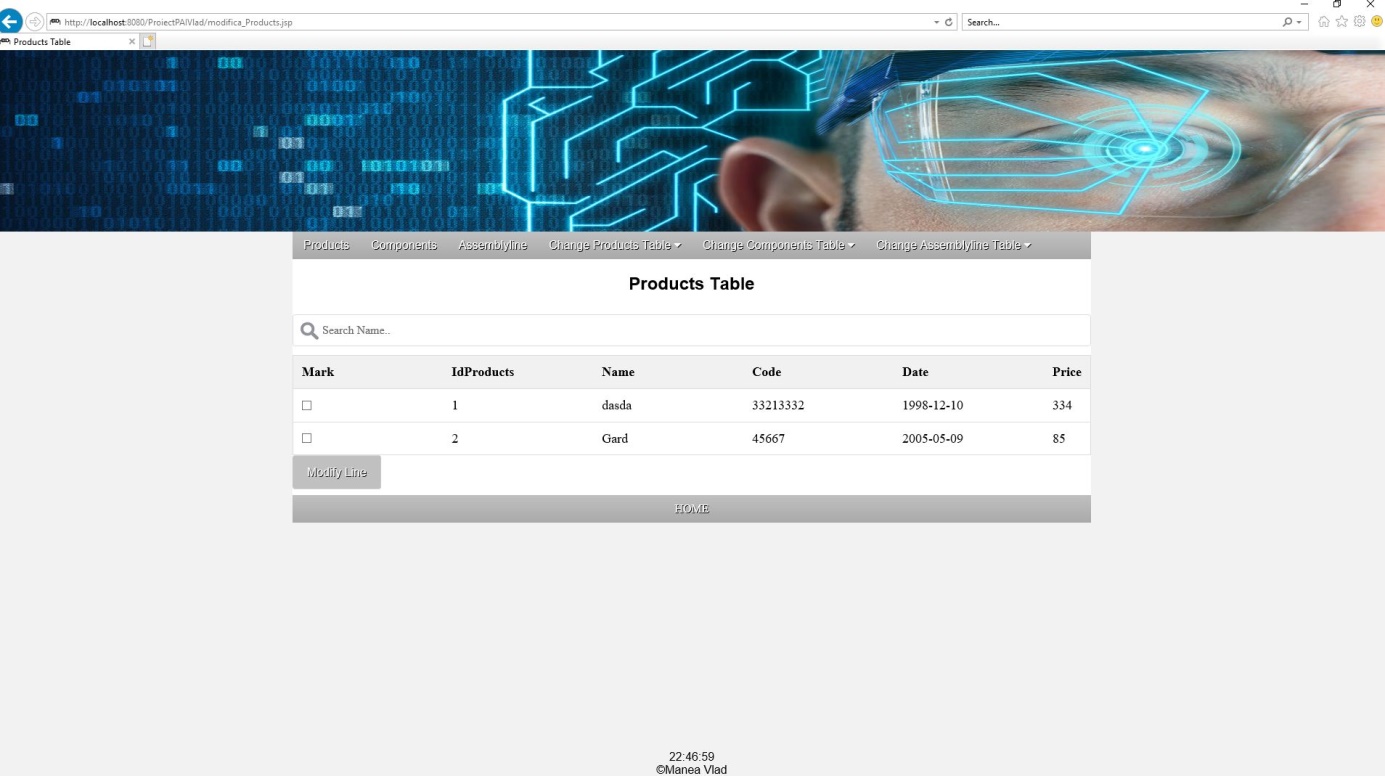
In pagina “m1\_Products.jsp” sunt scrise modificarile pe care vrem sa le aducem liniei alese in “modifica\_Products.jsp”.



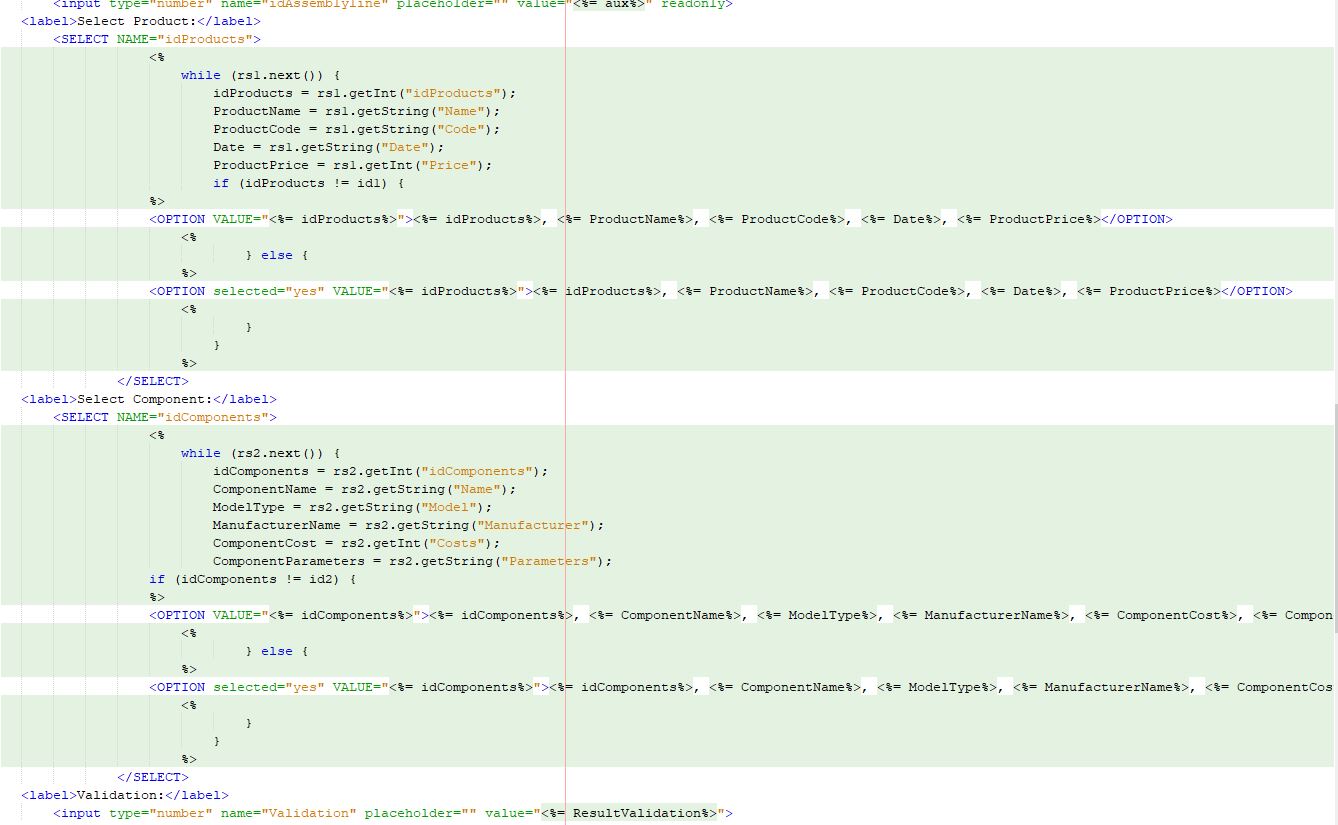


In pagina “m2\_Products.jsp” sunt aplicate modificarile scrise si ofera un raspuns daca modificarile au fost facute cu success.





Aceleasi pagini sunt create si pentru tabelele Components si Assemblyline, doar ca in paginile tabelei Assemblyline se vor interoga tabelele Products si Components pentru a folosi datele acestora, este necesar acest procedeu intrucat tabela Assemblyline este un tabel de legatura.In paginile “nou\_Assemblyline.jsp” si “m1\_Assemblyline” se poate observa interogarea tabelelor Products si Components.



Toata partea de design al paginilor este facuta in “stylesheet.css”.

body {

background: #f2f2f2;

}

header {

text-align: center;

background: #ffffff;

}

#container {

width: 1100px;

margin: 0 auto;

background: #ffffff;

}

nav {

overflow: hidden;

background-color: #C0C0C0;

background: linear-gradient(#C0C0C0,#A9A9A9);

font-family: Arial, Helvetica, sans-serif;

}

nav a {

float: left;

background: linear-gradient(#C0C0C0,#A9A9A9);

padding: 10px 15px;

color: #ffffff;

text-shadow: 1px 1px #000;

text-decoration: none;

text-align: center;

}

.dropdown {

float: left;

overflow: hidden;

}

.dropdown .drop {

border: none;

margin: 0;

outline: none;

background: linear-gradient(#C0C0C0,#A9A9A9);

padding: 10px 15px;

color: #ffffff;

text-shadow: 1px 1px #000;

text-decoration: none;

text-align: center;

}

nav a:hover, .dropdown:hover .drop, .dropdwon-content a:hover, #home:hover{

background: #808080;

}

nav a:active, .dropdown:active .drop, #home:active{

position: relative;

top: 1px;

}

.dropdown-content {

display: none;

position: absolute;

min-width: 150px;

z-index: 1;

}

.dropdown-content a {

background-color: linear-gradient(#C0C0C0,#A9A9A9);

color: #ffffff;

text-shadow: 1px 1px #000;

text-decoration: none;

text-align: center;

padding: 10px 15px;

float: none;

display: block;

}

.dropdown:hover .dropdown-content {

display: block;

}

h1,p {

margin: 0;

padding: 0;

outline: none;

border: none;

}

input[type=text], input[type=number], input[type=date], select{

width: 100%;

display: inline-block;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

padding: 12px 20px;

margin: 8px 0;

}

input[type=submit] {

background-color: #C0C0C0;

color: white;

border: none;

border-radius: 4px;

cursor: pointer;

padding: 14px 20px;

font-family: Arial, Helvetica, sans-serif;

margin-bottom: 8px;

text-shadow: 1px 1px #000;

text-decoration: none;

text-align: center;

}

label {

font-family: Arial, Helvetica, sans-serif;

padding: 10px 15px;

margin: 8px 0;

}

form {

margin: 0;

padding: 0;

outline: none;

border: none;

}

#home {

float: left;

background: linear-gradient(#C0C0C0,#A9A9A9);

padding: 10px 0;

margin-bottom: 40px;

color: #ffffff;

text-shadow: 1px 1px #000;

text-decoration: none;

text-align: center;

width: 100%;

}

#searchBar {

background-image: url('searchbar.png');

background-position: 10px 10px;

background-repeat: no-repeat;

width: 100%;

font-size: 16px;

padding: 12px 20px 12px 40px;

border: 1px solid #ddd;

margin-bottom: 12px;

}

#myTable {

border-collapse: collapse;

width: 100%;

border: 1px solid #ddd;

font-size: 18px;

}

#myTable th, #myTable td {

text-align: left;

padding: 12px;

}

#myTable tr {

border-bottom: 1px solid #ddd;

}

#myTable tr.header, #myTable tr:hover {

background-color: #f1f1f1;

}

#footer {

position: fixed;

left: 0;

bottom: 0;

width: 100%;

background-color: #f2f2f2;

text-align: center;

font-size: 16px;

font-family: Arial, Helvetica, sans-serif;

}

#center {

margin-top: 8px;

text-align: center;

font-size: 16px;

font-family: Arial, Helvetica, sans-serif;

}

.title {

text-align: center;

font-family: Arial, Helvetica, sans-serif;

color: #000;

text-shadow: 1px 1px #fff;

}

Dezvoltarea unei aplicatii web cu ajutorul platformei Hibernate

Aplicatia este construita in NetBeans folosind librariile MySQL JDBC Driver, JSTL 1.2.1, Hibernate 4.3.x si conexiunea este facuta prin intermediul serverului GlasFish Server 4.1.1. Aplicatia este dezvoltata cu ajutorul tehnologiei Java Server Pages (JSP) si a framework-ului Hibernate care va permite accesul la baza de date.

Se genereaza automat fisierul de configurare “hibernate.cfg.xml”, iar eu modific codul pentru a crea o conexiune cu baza de date “factory”:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.hbm2ddl.auto">update</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.show\_sql">true</property>

<property name="hibernate.query.factory\_class">org.hibernate.hql.internal.classic.ClassicQueryTranslatorFactory</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/factory?zeroDateTimeBehavior=convertToNull</property>

<property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">root</property>

<mapping resource="pojo/Products.hbm.xml"/>

<mapping resource="pojo/Assemblyline.hbm.xml"/>

<mapping resource="pojo/Components.hbm.xml"/>

</session-factory>

</hibernate-configuration>

Se creaza fisierul “hibernate.reveng.xml” de tip Hibernate -> Hibernate Reverse Engineering Wizard si adaug toate tabelele bazei de date.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-reverse-engineering PUBLIC "-//Hibernate/Hibernate Reverse Engineering DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-reverse-engineering-3.0.dtd">

<hibernate-reverse-engineering>

<schema-selection match-catalog="factory"/>

<table-filter match-name="assemblyline"/>

<table-filter match-name="products"/>

<table-filter match-name="components"/>

</hibernate-reverse-engineering>

Pachetul pojoeste este generat automat fiind de tip Hibernate ->Hibernate Mapping Files and POJOs from Database si contine trei clase java in care se genereaza automat functiile de get si set pentru toate tabelele.Exemplu pentru tabela Products:

POJO Products

package pojo;

// Generated Jan 19, 2019 2:11:30 AM by Hibernate Tools 4.3.1

import java.util.Date;

import java.util.HashSet;

import java.util.Set;

/\*\*

\* Products generated by hbm2java

\*/

public class Products implements java.io.Serializable {

private Integer idProducts;

private String name;

private String code;

private Date date;

private int price;

private Set<Assemblyline> assemblylines = new HashSet<Assemblyline>(0);

public Products() {

}

public Products(String name, String code, int price) {

this.name = name;

this.code = code;

this.price = price;

}

public Products(String name, String code, Date date, int price, Set<Assemblyline> assemblylines) {

this.name = name;

this.code = code;

this.date = date;

this.price = price;

this.assemblylines = assemblylines;

}

public Integer getIdProducts() {

return this.idProducts;

}

public void setIdProducts(Integer idProducts) {

this.idProducts = idProducts;

}

public String getName() {

return this.name;

}

public void setName(String name) {

this.name = name;

}

public String getCode() {

return this.code;

}

public void setCode(String code) {

this.code = code;

}

public Date getDate() {

return this.date;

}

public void setDate(Date date) {

this.date = date;

}

public int getPrice() {

return this.price;

}

public void setPrice(int price) {

this.price = price;

}

public Set<Assemblyline> getAssemblylines() {

return this.assemblylines;

}

public void setAssemblylines(Set<Assemblyline> assemblylines) {

this.assemblylines = assemblylines;

}

}

In pachetul DAO sunt trei interfete java:”ProductsDAO”, “ComponentsDAO”, “AssemblylineDAO” care contin toate functiile prin care se va comunica cu baza de date. Exemplu pentru “ProductsDAO”:

DAO Products

package DAO;

import java.util.List;

import java.util.Date;

import pojo.Products;

public interface ProductsDAO {

public void adaugaProduct (Products product);

public List<Products> afiseazaProducts();

public void modificaProduct (int idProducts, String name, String code, Date date, Integer price);

public void stergeProduct (Products product);

}

In pachetul DAOImpl este fisierul “HibernateUtil.java” care are scopul de a accesa SessionFactory pentru a obtine un obiect de tip Session, aceasta clasa apeleaza functia configure() si incarca fisierul “hibernate.cfg.xml” si construieste SessionFactory, codul este generat automat. In pachetul DAOImpl mai sunt trei clase java in care sunt implementate functiile din interfetele din pachetul DAO. Exemplu pentru fisierul “ProductsDAOImpl.java”:

DAOImpl Products

package DAOImpl;

import java.util.List;

import java.util.ArrayList;

import javax.persistence.Query;

import org.hibernate.Session;

import org.hibernate.Transaction;

import pojo.Products;

import DAO.ProductsDAO;

import java.util.Date;

public class ProductsDAOImpl implements ProductsDAO{

@Override

public void adaugaProduct(Products product) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = session.beginTransaction();

session.save(product);

transaction.commit();

session.close();

}

@Override

public List<Products> afiseazaProducts() {

List<Products> listaProducts = new ArrayList();

Session session = HibernateUtil.getSessionFactory().openSession();

org.hibernate.Query query = session.createQuery("From Products");

listaProducts = query.list();

return listaProducts;

}

@Override

public void modificaProduct(int idProducts, String name, String code, Date date, Integer price) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = session.beginTransaction();

Products detaliiProducts = (Products) session.load(Products.class, idProducts);

detaliiProducts.setName(name);

detaliiProducts.setCode(code);

detaliiProducts.setDate(date);

detaliiProducts.setPrice(price);

session.update(detaliiProducts);

transaction.commit();

session.close();

}

@Override

public void stergeProduct(Products product) {

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction transaction = session.beginTransaction();

session.delete(product);

transaction.commit();

session.close();

}

}

In pachetul Controller cunt trei clase de tip servlet in care se preia cererea de la server si se folosesc functiile definite in DAOImpl si pojo pentru a raspunde serverului cu un rezultat, acesta fiind adaugare de element, afisare tabela, modificare tabela si stergere date din tabela.

Exemplu pentru clasa “ProductsController”:

Controller Products

package Controller;

import java.io.IOException;

import java.util.List;

import java.util.Date;

import java.util.ArrayList;

import java.text.DateFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import pojo.Products;

import DAOImpl.ProductsDAOImpl;

public class ProductsController extends HttpServlet {

Products product = new Products();

ProductsDAOImpl productDaoImpl = new ProductsDAOImpl();

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

if (request.getParameter("adaugaProduct") != null) {

String name = request.getParameter("name");

String code = request.getParameter("code");

DateFormat df = new SimpleDateFormat("yyyy-MM-dd");

Date date = null;

try {

date = df.parse(request.getParameter("date"));

} catch (ParseException e) {

e.printStackTrace();

}

Integer price = java.lang.Integer.parseInt(request.getParameter("price"));

product.setName(name);

product.setCode(code);

product.setDate(date);

product.setPrice(price);

productDaoImpl.adaugaProduct(product);

RequestDispatcher rd = request.getRequestDispatcher("adauga\_Product.jsp");

rd.forward(request, response);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

if (request.getParameter("afiseazaProducts") != null) {

List<Products> listaProducts = new ArrayList();

listaProducts = productDaoImpl.afiseazaProducts();

request.setAttribute("listaProducts", listaProducts);

RequestDispatcher rd = request.getRequestDispatcher("tabela\_Products.jsp");

rd.forward(request, response);

}

if (request.getParameter("modificaProduct") != null) {

int id1 = Integer.parseInt(request.getParameter("idProducts"));

String name = request.getParameter("name");

String code = request.getParameter("code");

DateFormat df = new SimpleDateFormat("yyyy-MM-dd");

Date date = null;

try {

date = df.parse(request.getParameter("date"));

} catch (ParseException e) {

e.printStackTrace();

}

Integer price = java.lang.Integer.parseInt(request.getParameter("price"));

productDaoImpl.modificaProduct(id1, name, code, date, price);

RequestDispatcher rd = request.getRequestDispatcher("adauga\_Product.jsp");

rd.forward(request, response);

}

if (request.getParameter("stergeProduct") != null) {

int id2 = Integer.parseInt(request.getParameter("idProducts"));

product.setIdProducts(id2);

productDaoImpl.stergeProduct(product);

RequestDispatcher rd = request.getRequestDispatcher("adauga\_Product.jsp");

rd.forward(request, response);

}

}

@Override

public String getServletInfo() {

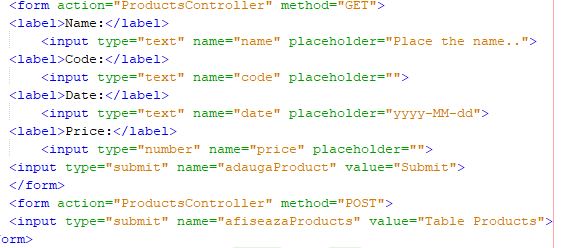
return "Short description";

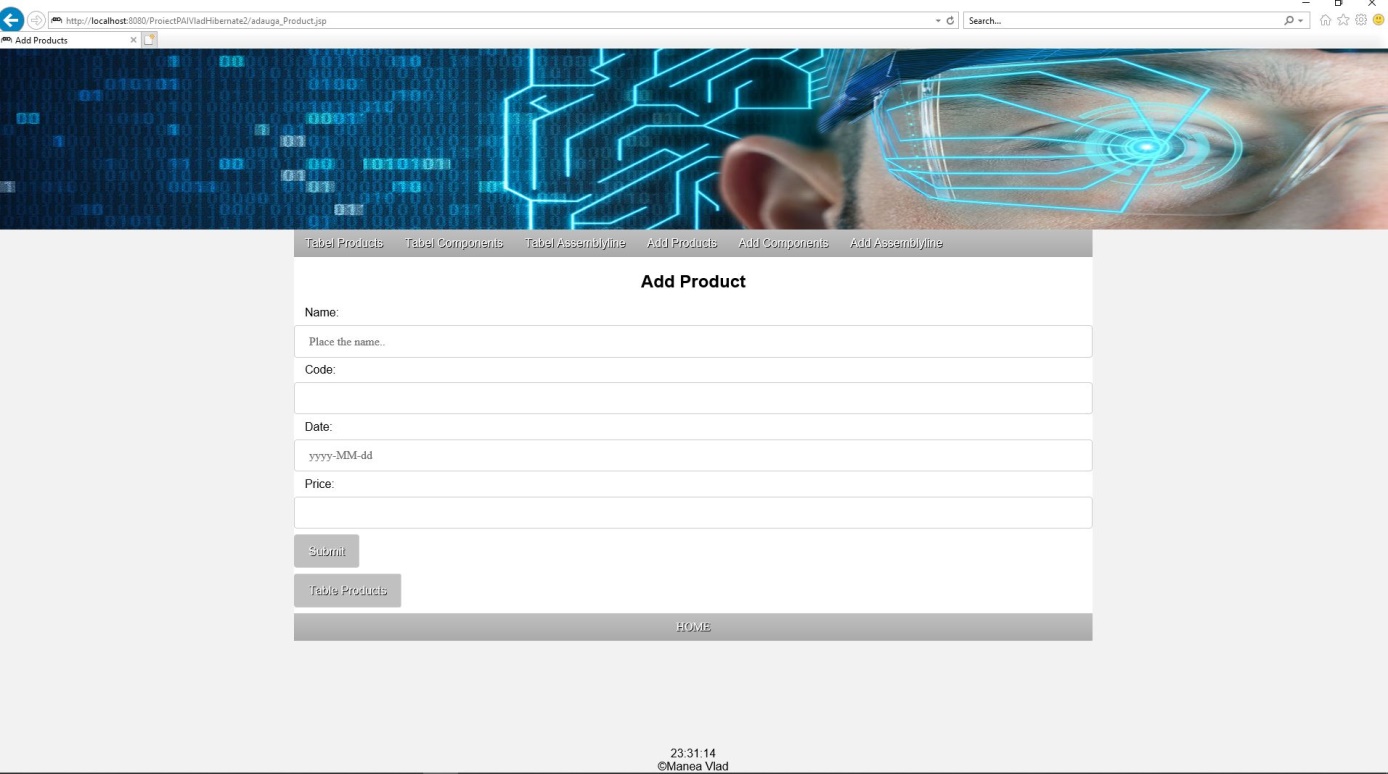
}

}

In fisierele de tip JSP “adauga\_Product.jsp”, “adauga\_Component.jsp”, “adauga\_Assemblyline.jsp” se actioneaza Controllerul pentru a adauga elemente in baza de date si pentru a afisa tabela. Exemplu pentru “adauga\_Products.jsp”:

Adauga Product

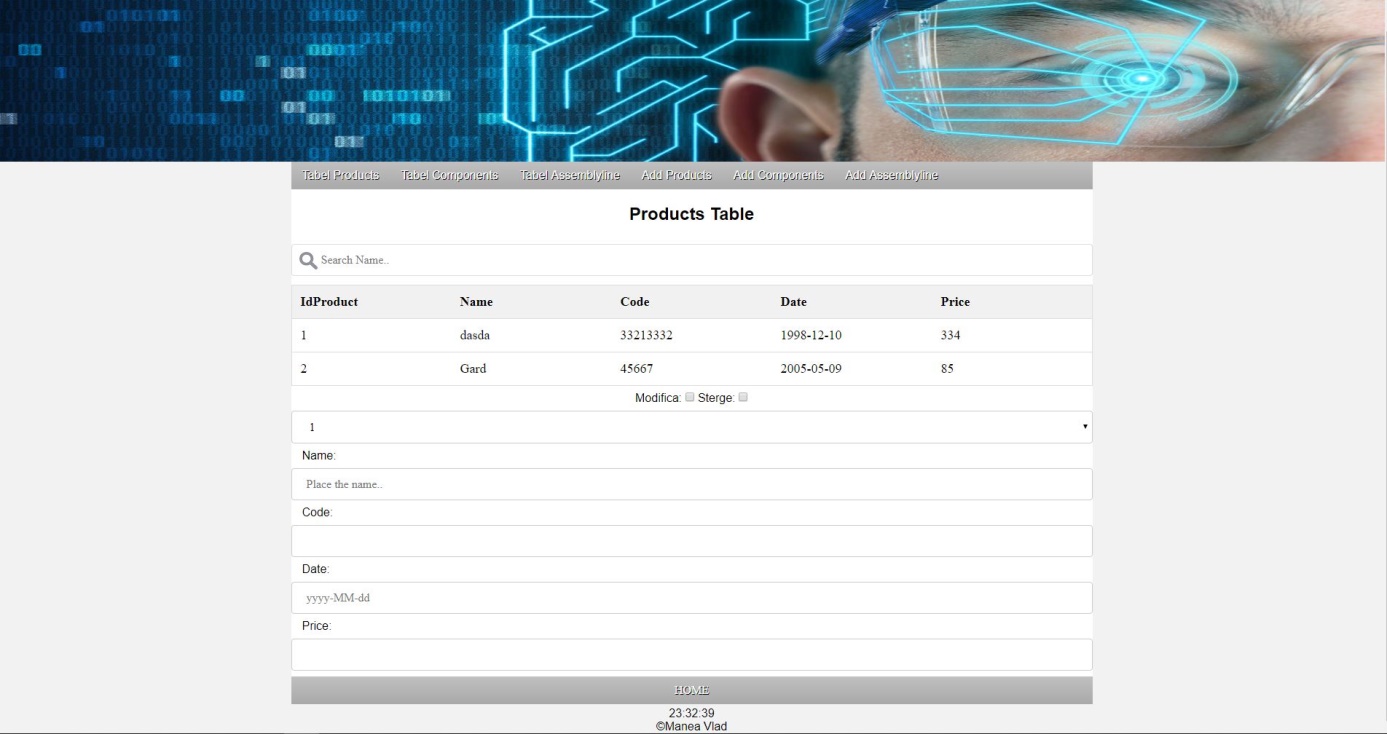




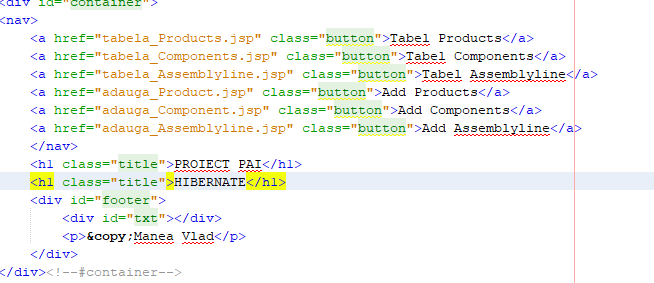
In fisierele de tip JSP “tabela\_Products.jsp”, “tabela\_Components.jsp”, “tabela\_Assemblyline.jsp” se actioneaza Controllerul pentru a modifica elemente sau pentru a sterge elemente din baza de date si pentru a afisa tabela. Exemplu pentru “tabela\_Products.jsp”:

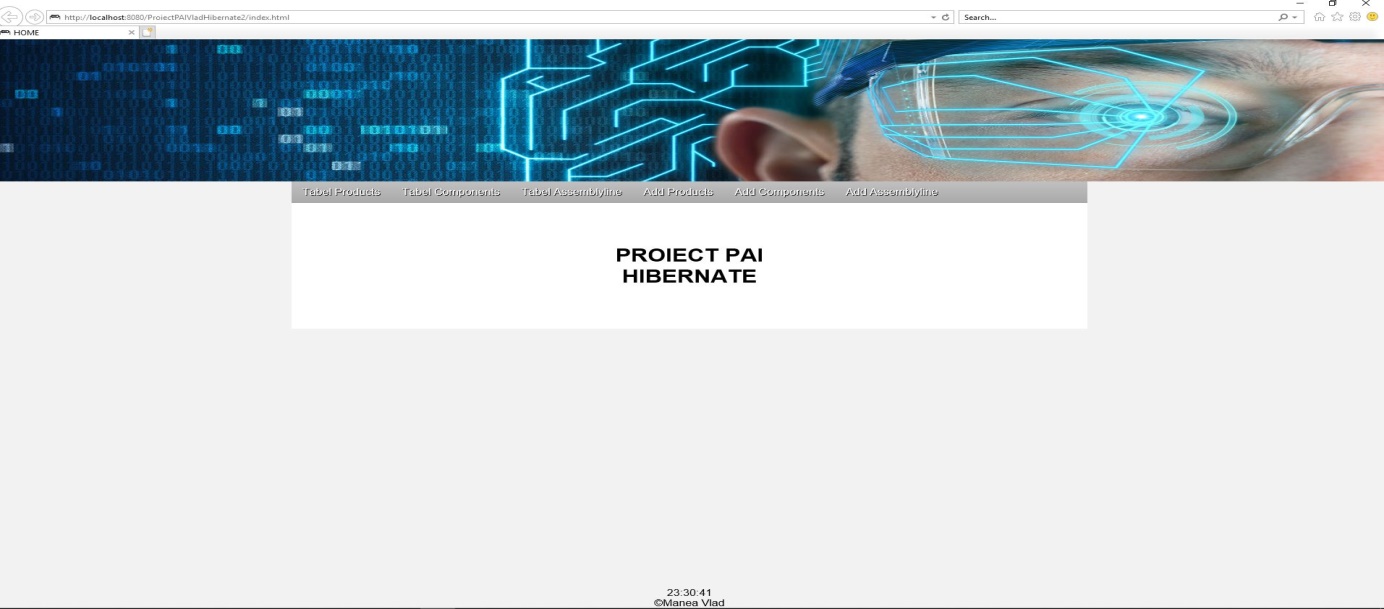
Tabela Product





Fisierul “index.html” care este creat automat este folosit ca o pagina principala din care se pot accesa toate celelalte pagini de tip JSP.





BIBLIOGRAFIE:

1. Cursul de PAI predat de dl. Valentin Pupezescu
2. <https://www.tutorialspoint.com/jsp/jsp_syntax.htm>
3. https://www.tutorialspoint.com/hibernate/