***Universitatea „POLITEHNICA” Bucuresti***

**FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI**

*Proiect*

***Programarea Interfețelor pentru Baze de Date***

***(Hibernate)***

*Mitrea Bogdan-Gabriel*

*Grupa 432Da*

*Prof. coordonator: Ș.l. Dr. Ing. Pupezescu Valentin*

*2023 – 2024*

**Cerință**

Creaţi două aplicaţii care să conţină o bază de date creată în sistemul de gestiune a bazelor de date MySql şi două interfeţe la aceasta (baza de date este comună). La crearea interfeţelor se vor folosi două tehnologii(la alegere - ex.: JSP, Hibernate, JPA, .NET, Python etc.).

**Introducere**

Am abordat tema gestionării magazinelor, a clienților și a tranzacțiilor. Am dezvoltat o interfață Web care interacționează cu o bază de date, stocând toate informațiile necesare într-un mod eficient. În această etapă a proiectului, am implementat funcționalități precum vizualizarea, adăugarea, modificarea și ștergerea datelor într-un mod elegant și intuitiv.

Un Sistem de Gestiune al Bazelor de Date (SGBD) reprezintă o aplicație sofisticată, responsabilă cu administrarea tuturor aspectelor legate de o bază de date.

Printre funcțiile esențiale ale unui SGBD se numără:

* facilitarea creării bazelor de date
* asigurarea securizată a stocării datelor
* traducerea algoritmilor în limbaj de programare
* furnizarea de comenzi detaliate pentru gestionarea datelor
* oferirea unei interfețe accesibile atât pentru programatori, cât și pentru utilizatorii neexperimentați.

În ceea ce privește infrastructura proiectului, s-a optat pentru MySQL Workbench 8.0, un sistem de gestionare a bazelor de date relaționale recunoscut, în principal, în contextul aplicațiilor online. Acesta se impune ca fiind cel mai popular SGBD open-source în prezent, reprezentând o componentă esențială a stivei LAMP (Linux, Apache, MySQL, PHP).

În cea de-a doua parte a proiectului, am ales tehnologia Hibernate pentru a implementa interfața cu baza de date. Hibernate este un framework Java care facilitează gestionarea relațională a bazelor de date într-un mod eficient și simplificat. Cu Hibernate, dezvoltatorii pot interacționa cu bazele de date folosind obiecte Java, eliminând nevoia de a scrie explicit cod SQL. Acesta oferă o abordare ORM (Object-Relational Mapping), transformând datele între structura obiectelor și cea relațională a bazelor de date, aducând astfel beneficii semnificative în dezvoltarea aplicațiilor Java bazate pe baze de date.Această decizie a contribuit la crearea unei interfețe eficiente și ușor de utilizat pentru utilizatori.

**Implementare**

Implementarea bazei de date a fost realizată prin intermediul MySQL Workbench, constând în două entități principale, anume tabelele "clients" și "shops". Pentru a gestiona relația de tip M:N între acestea, a fost concepută o entitate intermediară denumită "transaction".

**Structura coloanelor tabelelor:**

***Tabela clients***

* idClient (tip INT, cheie primară, Not Null, Unsigned, Auto-Increment)
* Nume (tip VARCHAR(45))
* Prenume (tip VARCHAR(45))
* Email (tip VARCHAR(45))
* Phone (tip VARCHAR(20))

***Tabela shops***

* idShop (tip INT, cheie primară, Not Null, Unsigned, Auto-Increment)
* NumeMagazin (tip VARCHAR(45))
* Localitate (tip VARCHAR(45))
* ContactEmail (tip VARCHAR(45))
* ContactPhone (tip VARCHAR(20))

***Tabela transaction***

* idTranzactie (tip INT, cheie primară, Not Null, Unsigned, Auto-Increment)
* idShop (tip INT)
* idClient (tip INT)
* PurchaseDate (tip DATE)
* TotalSpent (tip FLOAT)

Datorită asocierii de tip M:N, coloanele idClient și idShop au fost designate ca chei străine (foreign keys) în cadrul tabelei "transaction". Cheile primare corespunzătoare fiecărei tabele au fost configurate cu proprietățile Not Null (NN), Auto-Increment (AI), și Unsigned (UN). Diagrama aferentă tabelelor și structurii acestora este reprezentată în Fig. 1.

A diagram of a blockchain

Description automatically generated

Fig. 1 - Diagrama EER a tabelelor și conținutului acestora

În vederea implementării interfeței între utilizator și baza de date, am optat pentru tehnologia Hibernate.

**Pasi realizare cod:**

In prima faza, au fost create fisierele .xml „hibernate.cfg.xml” si „hibernate.reveng.xml” pentru conectarea platformei cu workbench-ul, apoi pentru conectarea platformei si cu cele trei tabele aferente.

Apoi, au fost create 4 pachete, speciale pentru a crea in acestea clase, interfete si fisiere java, pentru buna functionalitate a programului. Au fost create package-urile: pojo, DAO, DaoImpl, Controller.

Avem 3 interfete, cate una pentru fiecare tabela: clientsDao, shopsDao, transactionDao. Acestea contin metodele pentru adaugare, afisare, stergere din modificare campuri din baza de date. Interfetele sunt implementate de clasele clientsDaoImpl, shopsDaoImpl, transactionDaoImpl.

Pentru fiecare tabela avem cate o clasa controller, clientsController, shopsController, transactionController. Aceste clase deriva din clasa abstracta *HttpServlet* din package-ul *jakarta.servlet.http.HttpServlet* Aceasta clasa abstracta ne permite crearea unui *Servlet HTTP* care poate fi folosit pentru un website. Clasele definite de mine supraincarca metodele ***doGet*** (pentru request-uri de tip HTTP GET), ***doPost*** (pentru request-uri de tip HTTP POST) si ***getServletInfo*** (pe care servlet-ul il foloseste pentru a oferi informatii despre sine.

Clasa HibernateUtil este utila pentru generarea campului de tip Static SessionFactory care ne foloseste la toate operatiunile efectuate pe baza de date.

A screenshot of a computer

Description automatically generated

Fig. 2 – Structura finala a package-urilor

In pojo au fost create clasele si metodele aferente celor trei tabele, in DAO si DaoImpl s-au facut legaturile intre clase si noile interfete, iar in Controller apar fisierele care dau instructiuni catre fisierele java din webapp.

La fel s-a procedat si pentru celelalte clase, cu o atentie sporita asupra denumirilor, deoarece Hibernate este case-sensitive!

In final, Controllerul foloseste toate functiile implementate de DAOImpl, practic dand instructiuni pentru fiecare functie.

In cadrul acestui proiect am folosit si fisiere CSS, pentru a stiliza si a modela continutul paginii web.

**Interfata**

Prima pagină a interfeței *index.html*, dezvoltată exclusiv folosind limbajul de marcare HTML și limbajul descriptiv CSS, oferă acces la panoul de control al aplicației web. Acest panou permite vizualizarea, adăugarea si modificarea datelor din tabela pe care dorim să lucrăm.

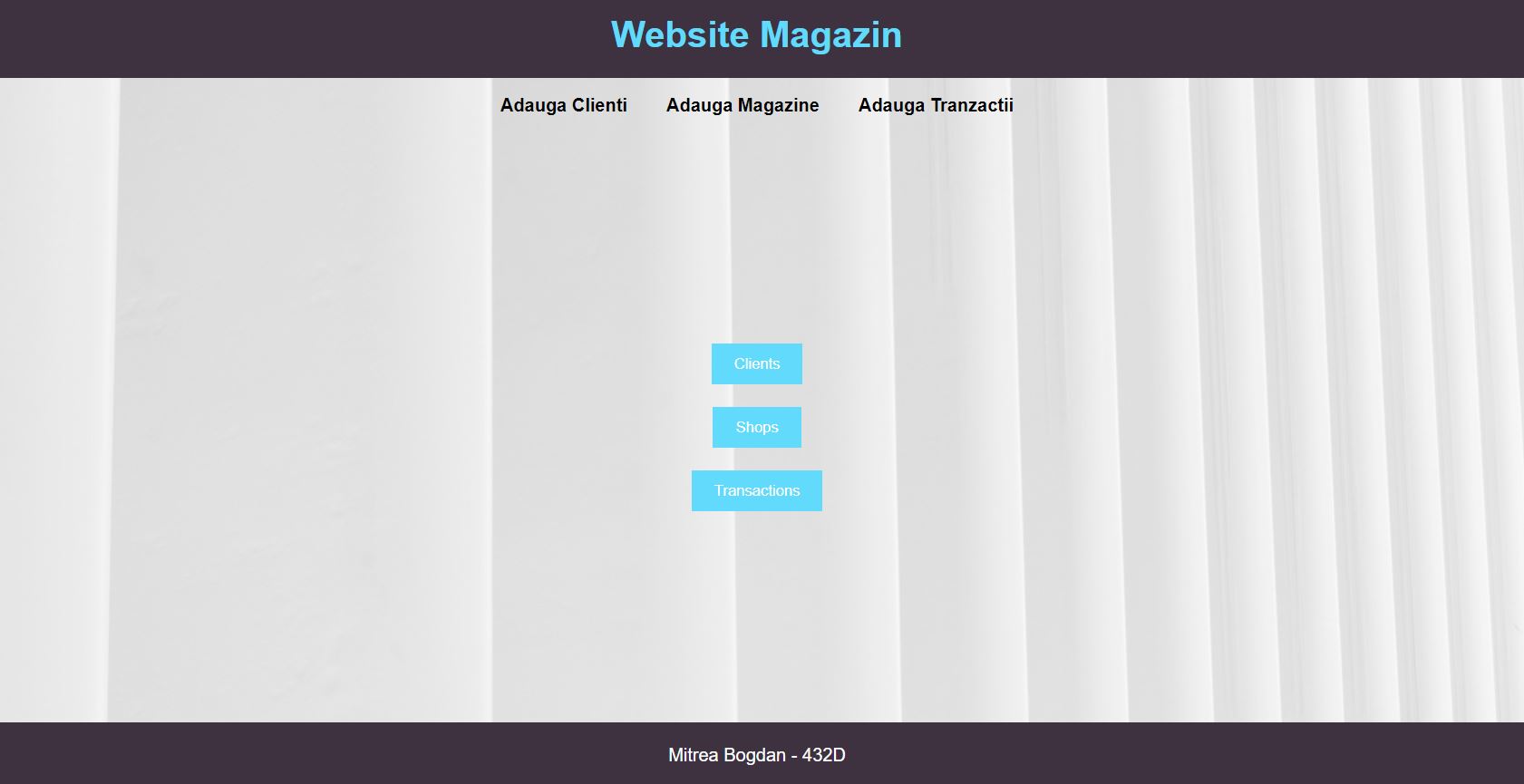
****

Fig. 3 – Pagina principala a interfetei

In aceasta pagina, utilizatorul beneficiază de posibilitatea de a accesa pagina corespunzătoare pentru vizualizarea, adaugarea și modificarea datelor asociate magazinelor, clienților și tranzactiilor, stocate în tabelele cu aceleași nume.

In cele ce urmeaza, funcționalitatea paginilor este similară în ceea ce privește operațiile legate de cele trei tabele: pentru a adauga o nouă înregistrare în tabela dorită (Clienti, Magazine sau Tranzactii), utilizatorul va apasa pe unul din cele trei butoane de sus.

Pentru a modifica oricare set de date din fiecare tabela, si pentru a vizualiza tabela efectiv, utilizatorul navigheaza pe unul din ultimele trei butoane.

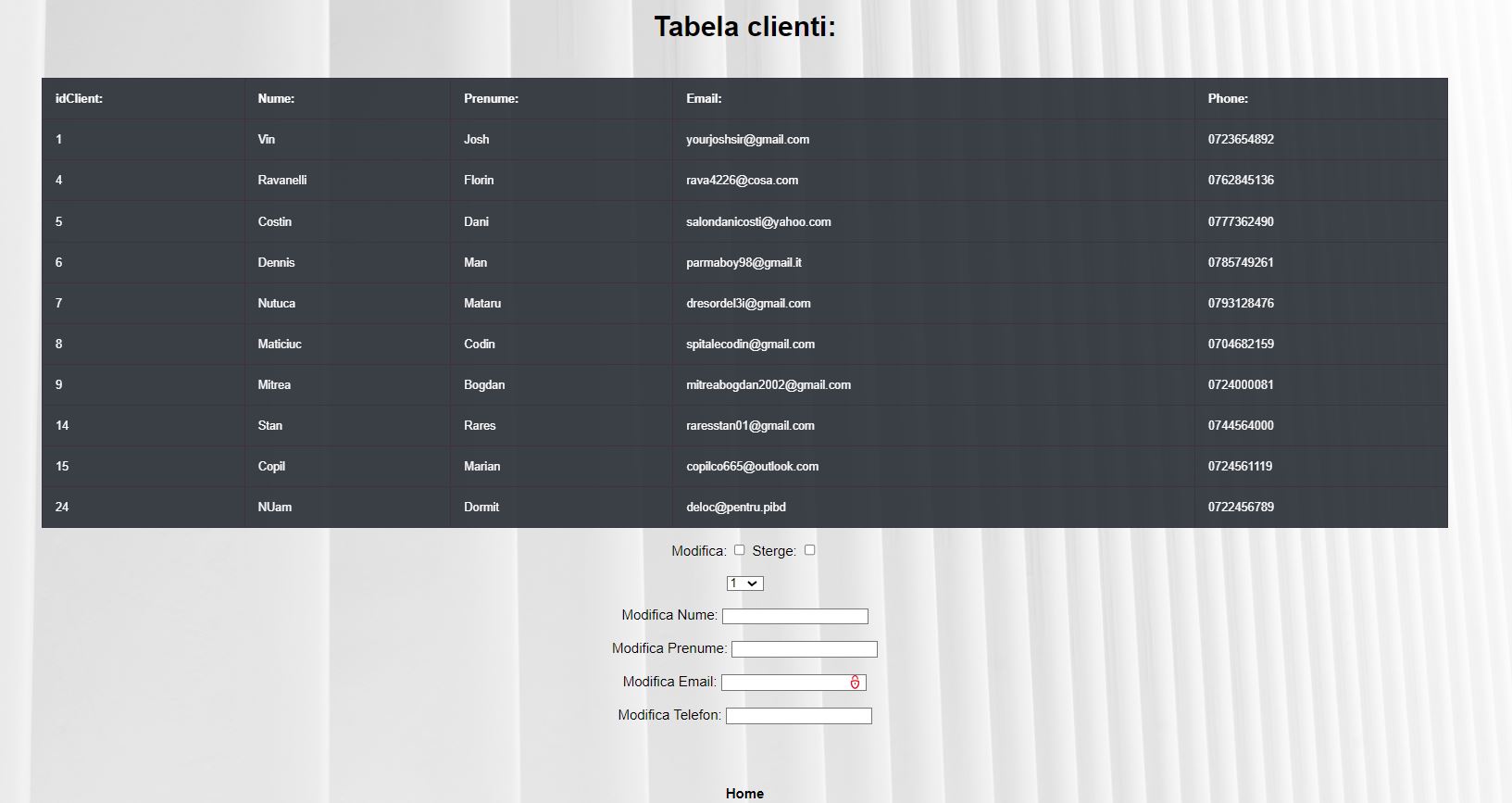


Fig. 4 – Tabela Clienti

Aici, avem posibilitatea de a vizualiza intreaga tabela „Clienti”. Mai jos apar doua casute, ce semnifica alegerea dintre a modifica sau a sterge o intrare, optiune urmata de cele 4 spatii libere de incarcare a noilor date. In functie de ce casuta s-a selectat, sub liniile completate va aparea butonul „Modifica”, sau butonul „Sterge”. Daca a fost aleasa casuta „Sterge”, nu mai este valabila decat selectarea id-ului dorit. In subsol, este present si butonul Home pentru a ne intoarce la pagina principala.

Pentru modificare se foloseste if-ul corespunzator din metoda doPost, in care se preiau campurile ce se doresc a fi modificate din formular si se apeleaza metoda modificaClients aferenta obiectului clientsDaoImpl. Aici se preia clientul existent din baza de date MySQL in obiectul detaliiClients si se modifica toate camprile cu datele din formular, apoi folosind session.update(detaliiClients) se actualizeaza in baza de date datele clientului respectiv.

Codurile aferente modificarilor sunt similare, principiul de baza fiind acelasi. Sunt mici diferente datorate numarului diferit de campuri din fiecare tabela si a tipului de date.

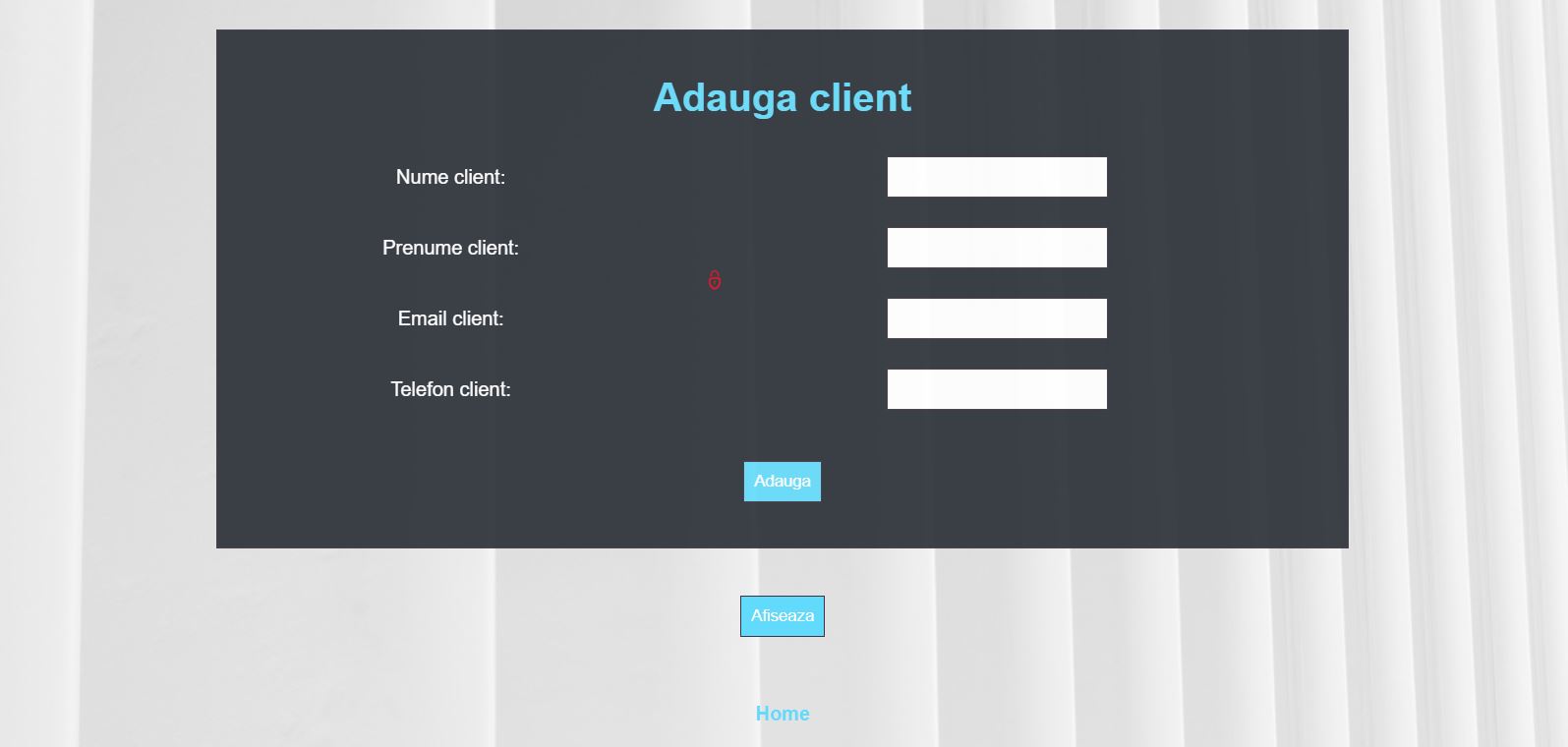


Fig. 5 – Adaugare Clienti

In aceasta fereastra putem insera un nou client in tabela. Dupa ce am introdus campurile corespunzatoare, se apasa pe campul „Adaugare” pentru a confirma adaugarea. Sub acesta exista si butonul „Afiseaza”, in cazul in care utilizatorul vrea sa aiba acces din aceasta fereastra catre tabela. De asemenea, utilizatorul se poate intoarce inapoi la Home.

Metoda adaugaClients, aferenta adaugarii unui client in baza de date:

public void adaugaClients(clients clients) {

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

Transaction trans = session.beginTransaction();

session.~~save~~(clients);

trans.commit();

session.close();

}

Metoda doGet din clasa clientsController, care se foloseste de metoda adaugaClients:

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

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

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

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

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

clients.setNume(Nume);

clients.setPrenume(Prenume);

clients.setEmail(Email);

clients.setPhone(Phone);

clientsDaoImpl.adaugaClients(clients);

RequestDispatcher rd = request.getRequestDispatcher("index.html");

rd.forward(request, response);

}

}

Paginile destinate vizualizării magazinelor și a tranzactiilor sunt similare celei de adăugare a clientilor, existând mici diferențe în funcție de numărul și tipul datelor fiecărei entități.

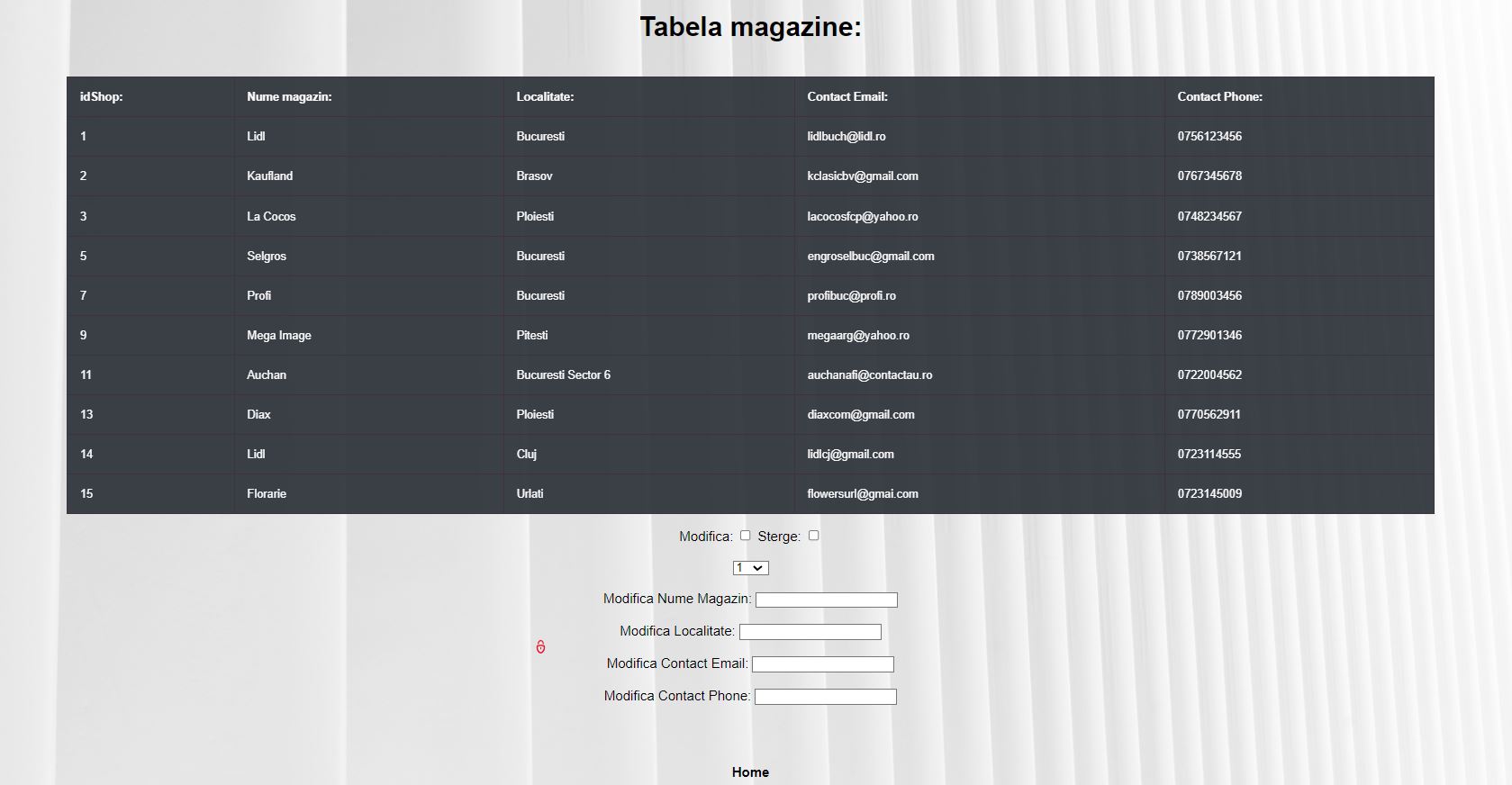


Fig. 6 - Tabela Magazine

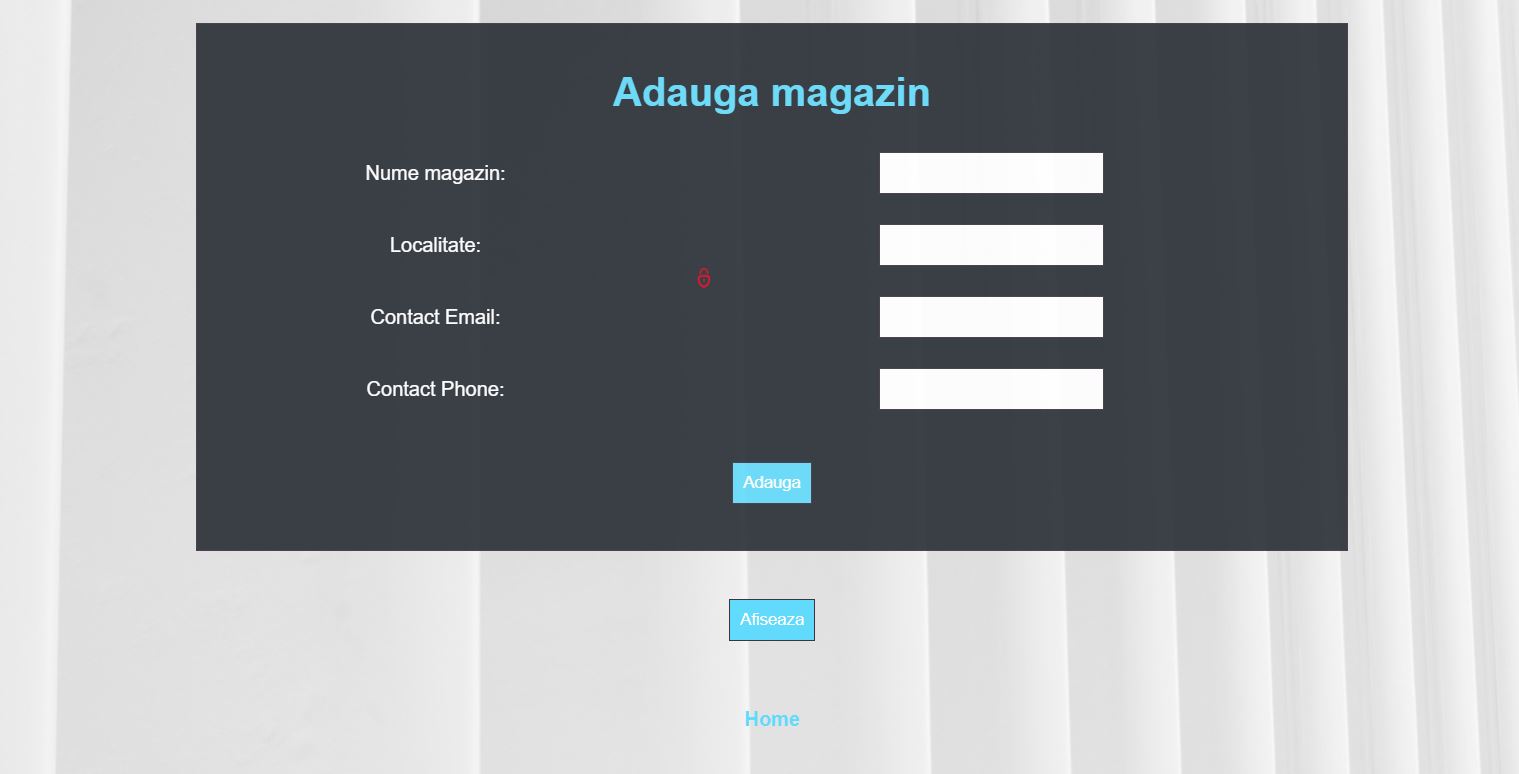


Fig. 7 - Adaugare Magazine

Adaugarea in tabela Magazine se face in acelasi mod, se folosesc clasele shopsDao si shopsController pentru adaugarea unui nou magazin in baza de date.

Tabela Tranzactii face legatura intre cele doua tabele anterioare, si vine cu doua noi atribute: data in care a fost facuta tranzactia (PurchaseDate), si totalul tranzactiei respective (TotalSpent).

Datorita faptului ca tabela Tranzactii este o tabela de legatura, in functia doPost din transactionController a trebuit sa facem rost de clientul si magazinul asociate unei tranzactii, folosindu-ne de metoda session#get si de ID-urile magazinului, respectiv clientului.

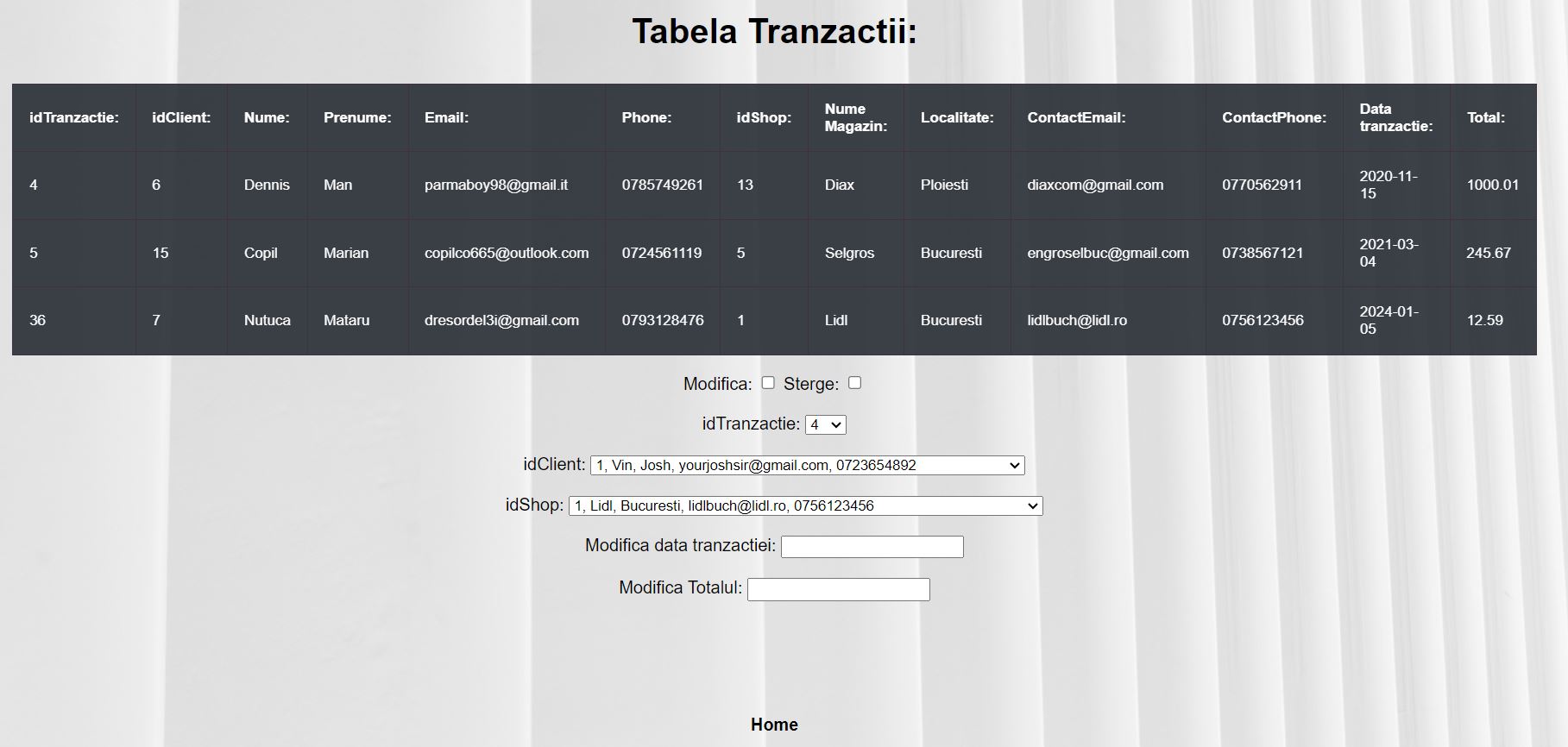


Fig. 8 - Tabela Tranzactii

Deoarece PurchaseDate este de tip string in formularul in care introduce utilizatorul o noua data a tranzactiei, am convertit variabila PurchaseDate in tipul Date cu formatul „yyyy-MM-dd”, cum este salvata in baza de date. Aceasta conversie poate esua, motiv pentru care am introdus conversia intr-un bloc de tip try-catch.

Adaugarea in tabela Tranzactii este similara, diferind prin faptul ca tabela Tranzactii este tabela de legatura si trebuie sa selectam clientul si magazinul aferenti unei tranzactii dintr-o lista deja definita.

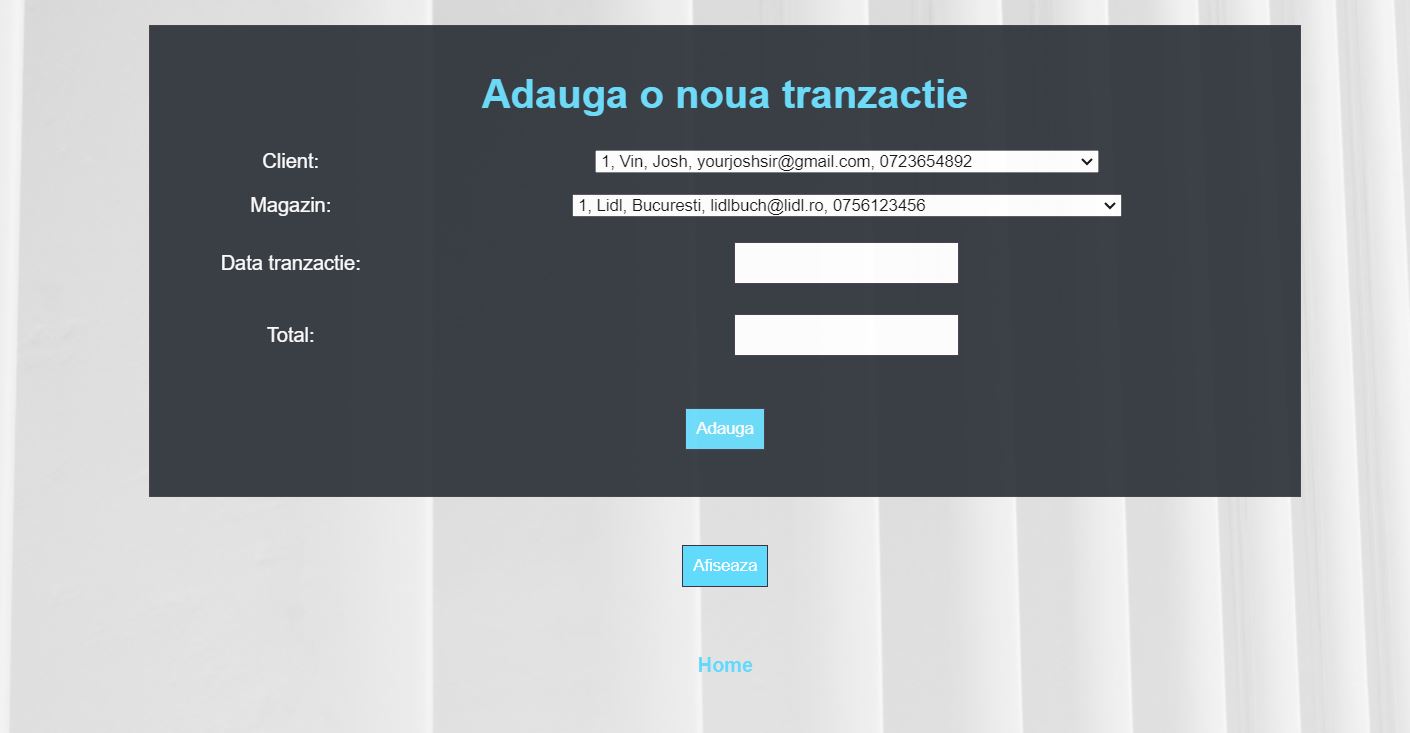


Fig. 9 - Adaugare Tranzactii

Am lucrat si la aspectul paginii, micsorand marimea fontului astfel incat tabelul sa incapa pe toata pagina, fiind foarte multe coloane.

Baza de date facuta in MySql este comuna celor 2 tehnologii, deci o modificare facuta in una din ele va fi vizibila si in  cealalta.

Pentru a imbunatati aspectul paginilor am adaugat elemente de programare HTML, Cascading Style Sheets (CSS) dar si mici bucati de cod javascript.

**Concluzii**

Cursul de *Programarea Interfețelor pentru Baze de Date* a contribuit semnificativ la extinderea perspectivelor asupra conceptului de baze de date și la dobândirea unei înțelegeri solide a conceptelor și tehnicilor esențiale pentru gestionarea și utilizarea eficientă a acestora. Am avut oportunitatea de a învăța cum să creăm și să administrăm o bază de date folosind Sistemul de Gestiune a Bazelor de Date (SGBD) MySQL, precum și modalitățile de conectare la un website utilizând platforma Hibernate.

Folosirea tehnologiei **Hibernate ORM** aduce mai multe beneficii: datele aplicatiei sunt disponibile si dupace aplicatia este oprita, Hibernate ORM ocupandu-se de persistenta datelor stocare in sisteme de gestiune a bazelor de date relationale. ORM (Object Relational Mapping) reprezintă o tehnică de programare de conversie a datelor stocate în sisteme de gestiune a bazelor de date relaţionale şi limbaje de programare obiect-orientate precum Java.

Prin intermediul interfeței dezvoltate, un administrator de date dispune de capacitatea de a gestiona fluxul clienților și comportamentul acestora în ceea ce privește acțiunile permise asupra contului lor de economii.

Interfața a fost concepută într-un mod intuitiv și clar, facilitând astfel operațiunile și pentru persoanele mai puțin familiarizate cu manipularea datelor propuse pentru stocare.

În ceea ce privește tehnologiile MySQL și platforma Hibernate, acestea au demonstrat a fi instrumente eficiente pentru crearea unei astfel de interfețe într-un mod elegant, oferind avantajul unei implementări ușoare și precise.

**Bibliografie**

[1] Prof. Pupezescu V., Curs de Programarea Interfețelor pentru Baze de Date 2023-2024

[2] \*\*\*, <https://repo.maven.apache.org/maven2/org/hibernate/> , accesat la data de 03.02.2024

[3] \*\*\*, <https://www.w3schools.com/> , accesat la data de 03.02.2024

[4] \*\*\*, <https://stackoverflow.com/questions/2288937/what-does-it-mean-the-serializable-class-does-not-declare-a-static-final-serial> , accesat la data de 03.02.2024

[5] \*\*\*, <https://itnext.io/whats-new-in-jakarta-persistence-3-1-by-examples-81b292e8b3a4> , accesat la data de 04.02.2024

[6] \*\*\*, <https://stackoverflow.com/questions/72016485/jakarta-persistence-persistenceexception-no-persistence-provider-for-entitymana> , accesat la data de 04.02.2024

[7] \*\*\*, <https://stackoverflow.com/questions/26860762/java-lang-noclassdeffounderror-could-not-initialize-class-xxx-xxx-xxx-hibernate> , accesat la data de 04.02.2024

[8] \*\*\*, <https://stackoverflow.com/questions/45957233/org-hibernate-query-cannot-be-converted-to-org-hibernate-query-query-javac-que> , accesat la data de 04.02.2024

[9] \*\*\*, <https://stackoverflow.com/questions/41098085/how-to-resolve-hibernate-cfg-xml-not-found-exception-when-session-factory-creat> , accesat la data de 05.02.2024

[10] \*\*\*, <https://stackoverflow.com/questions/27377711/maven-test-fails-with-noclassdeffounderror-for-antlr-class> , accesat la data de 05.02.2024

[11] \*\*\*, <https://docs.jboss.org/hibernate/orm/3.2/api/org/hibernate/Query.html> , accesat la data de 05.02.2024

[12] \*\*\*, <https://stackoverflow.com/questions/15022438/hibernate-configuration-troubles> , accesat la data de 05.02.2024