Moise Andrei

432A

PROGRAMAREA APLICATIVA A INTERFETELOR

ETTI-UPB

Cuprins

1. Introducere………………………………………2
2. Hibernate…………………………………….…..7
3. Prezentarea codului…………………………………………….11
4. Bibliografie……………………………………….17

CAPITOLUL 1

INTRODUCERE

* 1. Despre Proiectarea Interfetelor

Cand vine vorba de dezvoltare software, partea cea mai dificila, si cea mai consumatoare de timp este proiectarea si programarea interfetei cu utilizatorul(UI). Potrivit unor studii recente, programarea interfetei utilizator reprezinta undeva intre 30 si 90% din totalul liniilor de cod scrise pentru un software.[1]

Prima si probabil, cea mai importanta regula a proiectarii unei interfete este consistenta. Consistenta reprezinta una dintre cele mai de baza asteptari ale utilizatorului final, iar lipsa consistentei rezulta doar in frustrare in randul utilizatorilor care vor da click ca nebunii peste tot incercand sa gaseasca butonul de Ok, de exemplu. Un astfel de soft nu poate decat sa ajunga o relicva de care isi va aduce aminte doar Wikipedia. Pana la urma, cine ar vrea sa foloseasca un soft in care nu poate sa efectueze nici cele mai de baza operatii?[1]

A doua regula a proiectarii unei interfete reusite se refera la folosirea asa-numitelor „metafore” si „affordances”. O metafora reprezinta un mod de a crea, in mintea utilizatorului, o conexiune intre o actiune realizata in software si un obiect/actiune din lumea reala, pe care utilizatorul este mult mai probabil sa o recunoasca.[1]

Un exemplu foarte bun de metafora este butonul de Zoom, care se gaseste in aproape orice software. Daca butonul are si o iconita atasata, acea iconita va fi intotdeauna o lupa. Astfel, utilizatorul care doreste sa mareasca o anumita portiune dintr-o fotografie, de exemplu, nu va avea probleme in a folosi butonasul cu lupa, fara sa ii fie frica de modificarea dimensiunilor imaginii, deoarece stie ca lupa nu face asta in realitate. Un exemplu de „affordance” este coltul dreapta-jos al ferestrelor de Windows, care arata ca si cum ar fi facute special pentru aderenta, sugerand utilizatorului ca daca trage cu mouse-ul de el va redimensiona fereastra.[1]

Probabil cea mai cunoscuta metafora, desi putini realizeaza asta, este chiar Desktop-ul din Windows. Organizat cu fisiere si dosare, desktop-ul „simuleaza” biroul unei persoane. Iar utilizatorilor li s-a parut cat se poate de normal sa mute fisierele si sa le organizeze in foldere, deoarece asta faceau si in realitate pe birourile lor.[1]

Cu toate aceastea, o metafora prost aleasa poate crea atat confuzie, cat si frustrare. Ce s-ar intampla daca butonul de Zoom mentionat mai devreme ar mari efectiv dimensiunea imaginii? Sau daca butonul de selectare dintr-un program de editare de imagini ar actiona pe post de buton de decupare? Ne alegem cu un utilizator confuz care, cel mai probabil, nu isi va da seama ce a facut, si, mai ales, cum sa repare greseala(in cazul de fata un simplu Ctrl-Z este de ajuns, insa in alte cazuri aceasta optiune poate sa nu fie disponibila). Sansele sunt ca utilizatorul sa paraseasca exasperat aplicatia, si sa nu o mai foloseasca niciodata, pe motiv ca „nu merge, are buguri”.[1]

Ceea ce ne aduce la ultimul aspect al discutiei, aspect care e strans legat de primele doua: interfata unui software trebuie sa faca exact ce se asteapta utilizatorul sa faca. In exemplul de mai sus, in cele doua situatii nu se poate vorbi de bug-uri, ci de functionalitati pe care le are orice program decent de editare de imagini(mai putin Paint), doar ca metaforele alese pentru acele functionalitati erau total gresite. Daca metaforele ar fi fost alese astfel incat sa faca ceea ce utilizatorul se asteapta sa faca, tanti Lili, care are 80 de ani si nu isi recunoaste nepotii in poze decat daca fata lor apare pe jumatate de ecran, ar fi fost mai fericita. La fel si Mihai, care ar fi gasit butonul de Crop prin care si-ar fi putut elimina sotia din pozele de vacanta pe care i le va trimite secretarei. Un alt exemplu(din realitate) este comportamentul ferestrelor pe Windows vs. Mac. Pe Windows, clickul pe coltul din dreapta jos al ferestrei va redimensiona fereastra, in timp ce pe Mac acelasi click va muta fereasta. Acesta este un exemplu de comportament diferit, care, desi nu este gresit, va parea neintuitiv unui utilizator ce a fost obisnuit cu un sistem de operare si incearca sa treaca la unul nou.[1]

Cand vine vorba de proiectarea unei interfete, este foarte bine sa ne imaginam ce fel de utilizatori o vor folosi, si sa incercam sa empatizam cu ei. Da, poate eu stiu sa scriu ls –la | grep fisier.doc pentru a gasi un fisier intr-un folder suprapopulat, dar sunt foarte sigur ca secretara sefului nu stie. Ea prefera casuta de search. Treaba interfetei e sa simplifice lucrurile fata de vechea linie de comanda, si acest lucru este util chiar si utilizatorilor avansati.

Acest proiect propune dezvoltarea unor interfete pentru anumite baze de date in care un utilizator poate usor accesa continutul acesteia.[1]

* 1. MySQL

MySQL este un sistem de gestiune a bazelor de date relaționale, produs de compania suedeza MySQL AB și distribuit sub Licența Publică Generală GNU. Este cel mai popular SGBD open-source la ora actuală, fiind o componentă cheie a stivei LAMP (Linux, Apache, MySQL, PHP).[3][10][11]

Deși este folosit foarte des împreună cu limbajul de programare PHP, cu MySQL se pot construi aplicații în orice limbaj major. Există multe scheme API disponibile pentru MySQL ce permit scrierea aplicațiilor în numeroase limbaje de programare pentru accesarea bazelor de date MySQL, cum ar fi: C, C++, C#, Java, Perl, PHP, Python, FreeBasic, etc., fiecare dintre acestea folosind un tip specific API. O interfață de tip ODBC denumită MyODBC permite altor limbaje de programare ce folosesc această interfață, să interacționeze cu bazele de date MySQL cum ar fi ASP sau Visual Basic. În sprijinul acestor limbaje de programare, unele companii produc componente de tip COM/COM+ sau .NET (pentru Windows) prin intermediul cărora respectivele limbaje să poată folosi acest SGBD mult mai ușor decât prin intermediul sistemului ODBC. Aceste componente pot fi gratuite (ca de exemplu MyVBQL) sau comerciale. LAMP cuprinde și MySQL Licența GNU GPL nu permite încorporarea MySQL în softuri comerciale; cei care doresc să facă acest lucru pot achiziționa, contra cost, o licență comercială de la compania producătoare, MySQL AB.[3][12][14][8]

MySQL este componentă integrată a platformelor LAMP sau WAMP (Linux/Windows-Apache-MySQL-PHP/Perl/Python). Popularitatea sa ca aplicație web este strâns legată de cea a PHP-ului care este adesea combinat cu MySQL și denumit Duo-ul Dinamic. În multe cărți de specialitate este precizat faptul ca MySQL este mult mai ușor de invățat și folosit decât multe din aplicațiile de gestiune a bazelor de date, ca exemplu comanda de ieșire fiind una simplă și evidentă: „exit” sau „quit”. Pentru a administra bazele de date MySQL se poate folosi modul linie de comandă sau, prin descărcare de pe internet, o interfață grafică: MySQL Administrator și MySQL Query Browser. Un alt instrument de management al acestor baze de date este aplicația gratuită, scrisă în PHP, phpMyAdmin.[3][14]

MySQL poate fi rulat pe multe dintre platformele software existente: AIX, FreeBSD, GNU/Linux, Mac OS X, NetBSD, Solaris, SunOS, Windows 9x/NT/2000/XP/Vista.[3]

In Figura 1 sunt prezentate tabelele create in baza de date. Am avut de realizat o baza de date ce contine tabele propuse la care am avut de adaugat o tabela de lagatura de tipul M:N.[4]

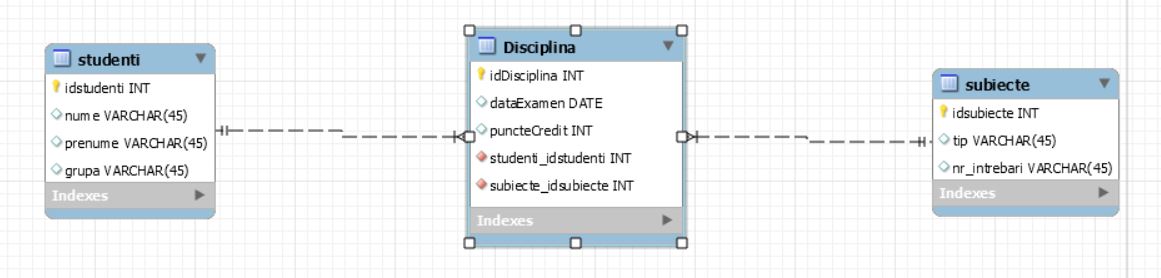


Figura 1.1 – Diagrama proiectului

Ce sunt relatiile de tip many-to-any?

Aceste tipuri de relaţii apar în prima fază a proiectării bazei de date, însă ele trebuie să fie ulterior eliminate. Figura 1.2 prezintă câteva exemple de relaţii many-to-many. La punctul b am considerat că un curs poate apărea pe oferta de cursuri a unei facultăţi, însă poate să nu fie aleasă de nici un student de aceea un curs poate fi urmat de unul sau mai mulţi studenţi. Invers, este posibil ca un student să fi terminat studiile şi să se pregătească pentru susţinerea examenului de licenţă şi de aceea el nu mai frecventează nici un curs. La punctul c, un profesor angajat al unei şcoli trebuie să predea cel puţin o disciplină. Iar o disciplină din planul de învăţământ trebuie să fie predată de cel puţin un profesor.[3][2][4]

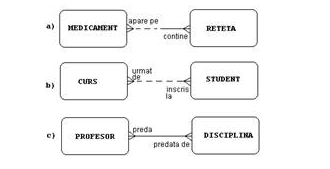


Figura 1.2 – Relații many-to-many

In figura 1.3 se poate gasit codul asociat bazei de date create in programul MySQL Workbench. Aici se pot observa cele 3 tabele create, in special ne intereseaza cea de legatura.[4]





Figura 1.3 – Baza de date create in Workbench

CAPITOLUL 2

HIBERNATE

## Sisteme de Tip ORM (Object-Relational Mapping)

Termenul de ORM (Object-Relational Mapping) este folosit pentru a descrie o tehnică de programare prin care sunt transformate informațiile între formatul în care sunt stocate în bazele de date relaționale și formatul în care sunt reprezentate în limbajele de programare orientate-obiect.[16]

O soluție ORM este formată din mai multe componente:

* interfață de programare care să ofere funcționalități de tip CRUD (Create, Read, Update, Delete) pentru instanțele unei clase persistente;
* un limbaj (sau o interfață de programare) prin care să poată fi descrise interogări referitoare la clasele persistente și la diferitele proprietăți ale acestora;
* un mecanism flexibil de specificare a modului în care datele dintr-o formă de reprezentare pot fi convertite în cealaltă formă de reprezentare;
* tehnică prin care să se poată utiliza obiectele tranzacționale astfel încât să se realizeze diferite funcții de optimizare cum ar fi evitarea scrierilor murdare, a citirilor nerepetabile sau a citirilor fantomă, precum și a preluării asocierilor evaluate leneș (eng. lazy association fetching).

Necesitatea de a folosi un sistem ORM este dată de incompatibilitățile între modelul de date relațional (în care informațiile sunt reprezentate într-o formă tabulară) și modelul de date obiectual (în care informațiile sunt stocate sub forma unui graf de obiecte interconectate). Problemele pe care le adresează un sistem de mapare de tip ORM sunt legate de:

* granularitate - modelul de date obiectual conține de regulă mai multe entități decât modelul de date relațional;
* moștenire - mecanismul de moștenire, descris de limbajele de programare orientate obiect, nu este suportat de cele mai multe dintre sistemele de gestiune pentru baze de date relaționale;
* identitate - noțiunea de similaritate în sisteme de gestiune pentru baze de date relaționale este descrisă prin intermediul cheii primare în timp ce modelul obiectual face distincția între identitate (verificarea fiind realizată la nivelul referințelor către obiecte) și egalitate (verificarea referindu-se la conținutul propriu-zis al obiectelor, adică al valorilor pe care le iau proprietățile claselor);
* asociere - în sistemele de gestiune pentru baze de date, referințele sunt reprezentate prin intermediul cheii străine în timp ce modelul obiectual utilizează referințele între instanțe ale claselor;
* navigare - modul de acces la informații este diferit în cele două mecanisme de reprezentare al datelor.

Sistemele de tip ORM folosesc un proces bazat pe mai multe etape în conversia dintre un obiect și o înregistrare la nivelul bazei de date, interacțiunea dintre acestea purtând numele de mapare obiectual-relațională:

1. etapa obiectului care încapsulează date (eng. object data) conține:

* obiecte POJO (Plain Oriented Java Objects), care descriu entitățile pentru care se asigură persistența;
* clase și interfețe de tip serviciu, care oferă diferite funcționalități (de tip CRUD) referitoare la entitățile persistente, reprezentând componentele care implementează logica aplicației;

1. etapa de asociere sau realizare a persistenței este formată din mai multe componente:

* furnizorul de servicii JPA reprezintă un produs specific care implementează interfața de programare JPA (Hibernate, EclipseLink, Spring Data JPA, TopLink, OpenJPA, DataNucleus);
* fișierul ce descrie mapările (ORM.xml) conține legăturile dintre proprietățile unui obiect POJO și atributele tabelei asociate din cadrul bazei de date;
* obiectul de încărcare JPA descrie o zonă de memorie în care pot fi stocate informațiile din sursa de date relaționale, putând fi integrată cu clasele și interfețele de tip serviciu pentru a putea accesa proprietățile unei entități persistente;
* tabela de obiecte este o locație temporară care poate stoca informații din baza de date și pe care sunt realizate toate interogările, înainte de a fi transmise mai departe către sursa de date;

1. etapa informațiilor relaționale conține sursa de date conectată la obiectul de logică a aplicației[16]

## 2.2Hibernate

Hibernate reprezintă o soluție de tip ORM pentru asigurarea persistenței și a interogării eficiente a informațiilor stocate în baze de date relaționale, prin intermediul limbajului de programare Java. Este un produs open-source ce realizează maparea dintre tabelele bazei de date și clase Java, asigurând astfel interacțiunea dintre sistemeul de gestiune pentru baze de date și serverul de aplicații ce implementează logica aplicației.[16]

Mecanismul prin care sunt reprezentate legăturile dintre baza de date și clasele Java este reprezentat de fișiere XML, astfel încât nu este necesară dezvoltarea de cod sursă pentru obținerea unei astfel de funcționalități. Astfel, în situația în care apare o schimbare la nivelul bazei de date sau la nivelul claselor Java, este suficient ca aceasta să fie realizată și la nivelul fișierului XML corespunzător pentru ca legătura dintre entități să fie menținută. API-ul pus la dispoziție atât pentru asigurarea persistenței cât și pentru interogarea diferitelor informații din baza de date este flexibil și scalabil, fiind accesibil tuturor programatorilor, indiferent de nivelul de pregătire. Se oferă o abstractizare a operațiilor de la nivelul sistemului de gestiune pentru baze de date, astfel încât detaliile de implementare de la nivelul acestuia nu trebuie să fie cunoscute, interacțiunea făcându-se la nivelul obiectelor Java ce corespund unor concepte specifice afacerii sau culturii organizației respective. Pot fi modelate astfel legături oricât de complexe dintre tabelele entitate de la nivelul bazei de date. Mai mult, accesul la date se realizează optimizat, folosindu-se diferite strategii inteligente pentru interogarea informațiilor, menținând în același timp ușurința în utilizare. De asemenea, poate fi utilizat independent de un server de aplicații.[16]

Hibernate suportă mai multe sisteme de gestiune pentru baze de date, cum ar fi Oracle, MySQL, PostgreSQL, Microsoft SQL Server, Sybase, Informix Dynamic Server, DB2, HSQL, precum și mai multe tehnologii cum ar fi Java Enterprise Edition, XDoclet Spring, plugin-uri Eclipse sau Maven.[16]

Arhitectura Hibernate este structurată pe mai multe niveluri, astfel încât să fie încapsulate cât mai multe dintre interfețele de programare pe care le utilizează, oferind aplicației servicii legate de obiectele persistente.[16]

1. Un obiect persistent reprezintă o entitate, reprezentată de regulă sub forma uneia (sau mai multor) tabele la nivelul bazei de date, pentru care se asigură mecanisme de stocare și de încărcare la nivelul aplicației Java relativ la sursa de date respectivă. Comunicația dintre aplicația Java și Hibernate este realizată prin intermediul serviciilor care furnizează obiecte persistente pe baza parametrilor furnizați (aceștia putând fi, la rândul lor, tot obiecte persistente).[16]

2. Structura Hibernate este reprezentată de mai multe clase:

* Configuration conține proprietățile din fișierele de configurare (hibernate.properties sau hibernate.cfg.xml) necesare pentru funcționarea Hibernate; o astfel de clasă are o instanță unică la nivelul aplicației Java care exploatează informații dintr-o sursă de date, obiectul având rolul de a specifica parametrii de conexiune la baza de date dar și de a realiza procesul de asociere dintre modelul relațional și modelul obiectual;
* SessionFactory reprezintă o clasă instanțiată pentru o anumită configurație Hibernate, furnizând (pentru toate firele de execuție de la nivelul aplicației) sesiuni prin care se oferă acces propriu-zis la baza de date; întrucât este un obiect care necesită multe resurse, este utilizată o instanță unică (pentru un anumit fișier de configurare), inițializată la un moment dat și refolosită pe parcusul ciclului de viață al aplicației Java;
* Session este utilizat pentru a obține o conexiune propriu-zisă la baza de date; întrucât este un obiect care necesită puține resurse, poate fi instanțiat de fiecare dată când este necesară interacțiunea cu sursa de date; funcționalitățile pe care le oferă includ stocarea și încărcarea unui obiect persistent; un astfel de obiect nu trebuie să fie utilizat mai mult decât este necesar, întrucât nu sunt sigure într-un mediu de execuție concurent;
* Transaction constă dintr-o serie de operații care trebuie să fie executate în mod atomic; o astfel de funcționalitate este oferită de alte interfețe de programare cu care Hibernate interacționează, nefiind obligatorie utilizarea acestei interfețe, utilizatorii putând să aleagă să implementeze propriul mecanism prin care să asigure un comportament tranzacțional;
* Query descrie interogări în limbajul SQL sau HQL (Hibernate Query Language) pentru a gestiona informațiile reprezentate la nivelul bazei de date; sunt implementate funcționalități pentru legarea parametrilor disponibili în momentul execuției aplicației sau pentru limitarea numărului de rezultate;
* Criteria specifică interogări într-un limbaj orientat pe obiecte, manipulând informațiile de la nivelul bazei de date.

3. Interacțiunea cu baza de date este realizată prin intermediul mai multor interfețe de programare:

* JDBC oferă acces la sursa de date prin intermediul unui driver de conectare, permițând ca interogarea datelor să se facă într-un mod similar ca în cazul exploatării sistemului de gestiune pentru baze de date, în limbajul nativ;
* Java Transaction API (JTA) și Java Naming and Directory Interface (JNDI) asigură integrarea cu servere de aplicații folosite de tehnologia Java Enterprise Edition.[16]

CAPITOLUL 3

PREZENTAREA CODULUI

## Tiparul MVC

Acesta este reprezentat de tiparul Model-Vizualizare-Controller. Acest tipar ce apare in Figura 3.1 este folosit sa separe nevoile aplicatiei.[4]

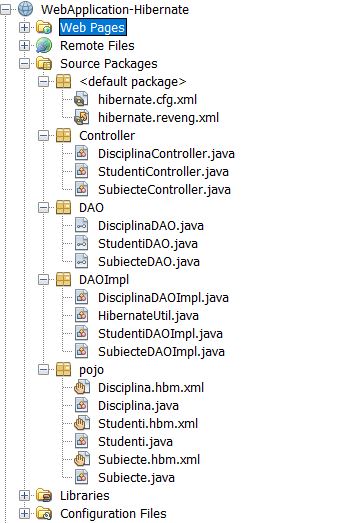


Figura 3.1 – Tiparul MVC

Model - reprezinta un obiect sau date de transfer de tip JAVA POJO. Este folosit si pentru a acutaliza controllerul daca datele sale se schimba.[4]

Vizualizarea - reprezinta partea de cod in care se realizaeaza vizualizarea datelor pe care modelul le contine.[4]

Controller – acesta actioneaza ca model SI ca vizualizare. El controleaza fluxul de de date in obiectul model si actualizeaza vizualizarea cand datele se schimba. Este realizat cate un model in parte pentru fiecare tabela. Controllerul intercepteaza cererea HTTP si returneaza raspunsul HTTP. In urmatoarea figura, Figura 3.2, este prezentat codul care face legatura intre tabela “subiecte” si aplicatie.[4]



Figura 3.2 – Codul ce face legatura intre tabela ”subiecte” si aplicatie

Codul a fost impartit in mai multe bucati deoarece fiecare indeplineste o functie separata:

* obiectul SessionFactory: obiectul de configurare este folosit pentru a crea un obiect de tip SessionFactory care, la randul sau, configureaza Hibernate(Figura 3.3) pentru aplicatie folosindu-se de fisierul de configurare si permite unui obiect Session sa fie instantiate. Este nevoie de un obiect SessionFactory pentru fiecare baza de date foolosind un fisier separate de configurare.
* obiectul de tranzactie: o tranzactie reprezinta o unitate de lucru cu baza de date si majoritatea bazelor de date relationale suporta aceasta functionalitate. Tranzactiile in Hibernate sunt facute de un manager de tranzactii.
* obiectul Query: obiectele Query folosesc string de tip SQL sau HQL pentru a primi date de la baza de date si sa creeze obiecte. O instant Query este folosita sa lege parametrii Query, limiteaza numarul de rezultate intoarce de Query si, la final, sa execute Query-ul.[4]

Mai jos in Figura 3.3 sunt atasate bucati din interfata creata:

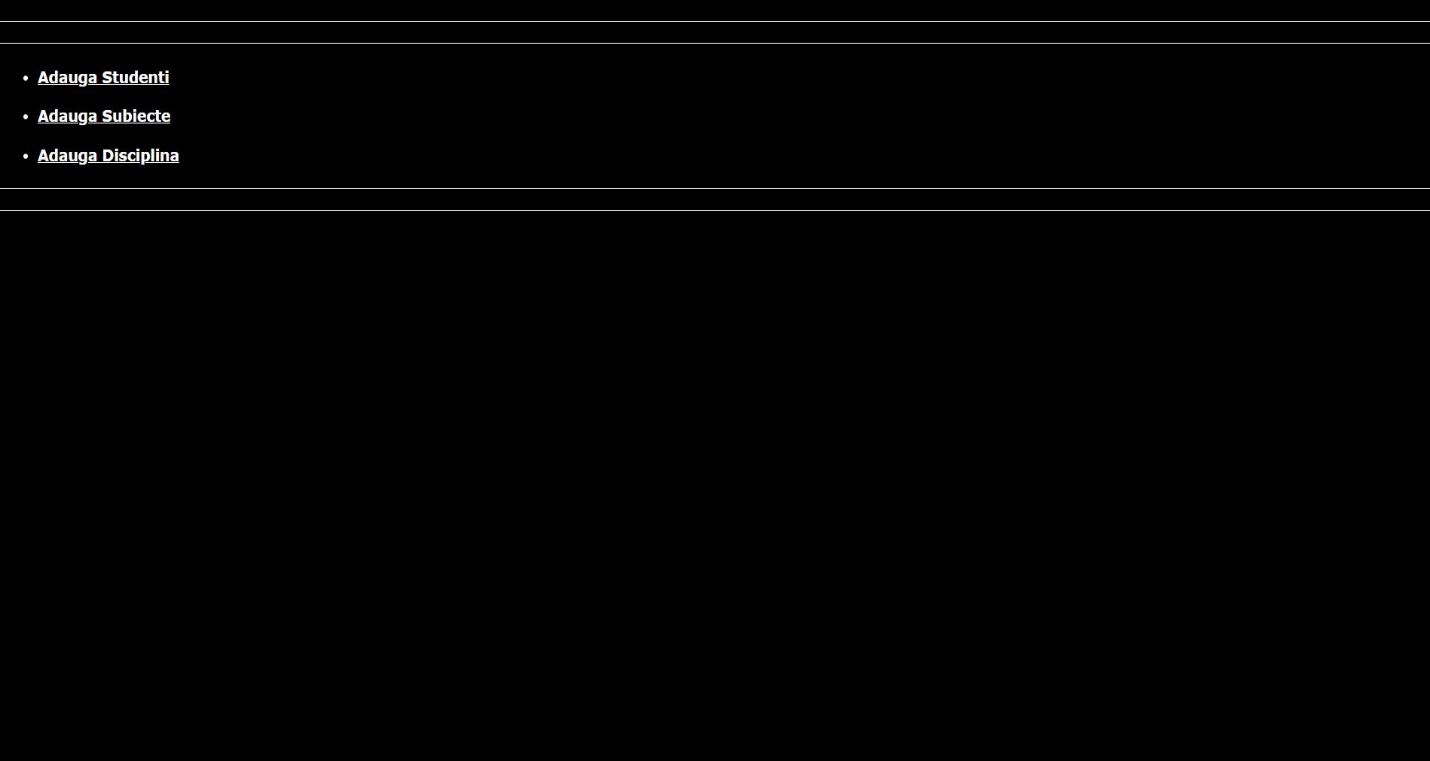


Figura 3.3 – Pagina principala



Figura 3.3 – Codul asociat paginei principale

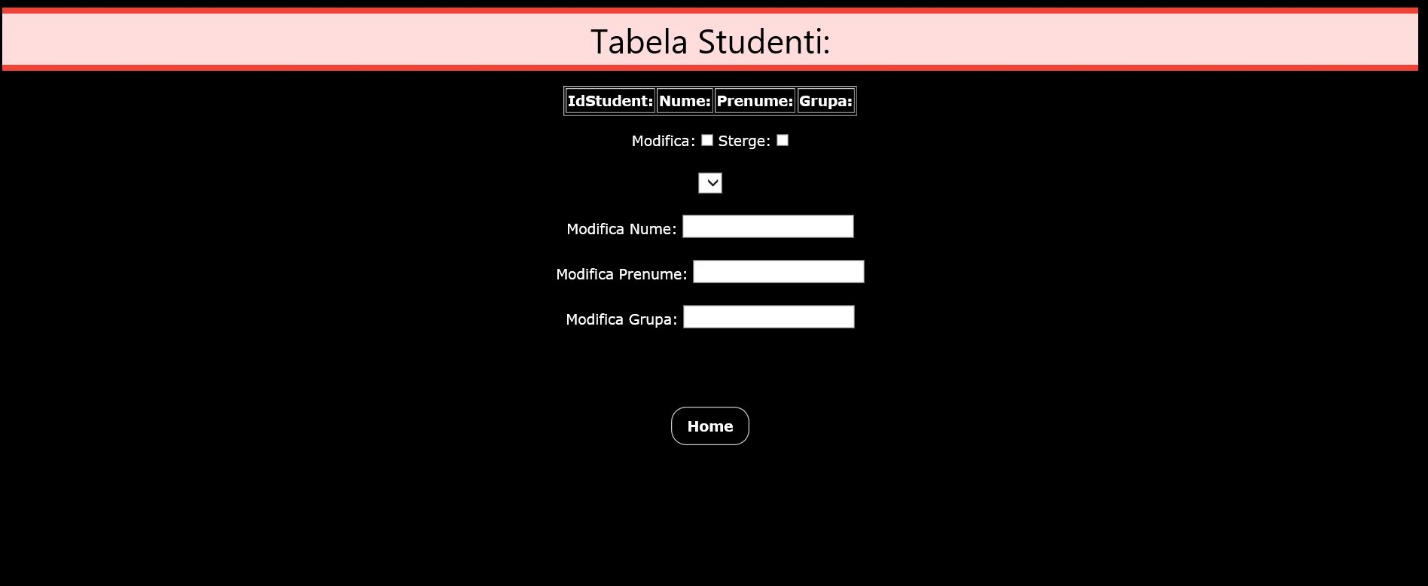


Figura 3.4 – Tabela “studenti”





Figura 3.5 – Codul asociat tebelei “studenti”



Figura 3.6 – Functia de adaugare student



Figura 3.6 – Codul asociat functiei de adaugare studenti

Bibliografie

1. http://www.ligasoftware.ro/blog/2011/03/despre-proiectarea-interfetelor
2. http://www.atestateinformatica.ro/tutoriale/relatii-intre-entitati
3. 3.
4. [https://ro.wikipedia.org/wiki/MySQL](https://ro.wikipedia.org/wiki/MySQL )
5. Curs Programare aplicativa a interfetelor, Valentin Pupezescu
6. Dan C. Cosma, "Programarea aplicatiilor distribuite", Editura de Vest, 2009, Timisoara, ISBN 978-973-36-0501-0.
7. http://control.aut.utcluj.ro/scd/lab6/tehnologiiWeb.htm
8. „Changes in MySQL 5.6.19”. MySQL 5.6 Reference Manual. Oracle. 30 mai 2014. Accesat la 30 mai 2014.
9. „Changes in Release 5.7.4 (Development)”. MySQL 5.7 Reference Manual. Oracle Corporation. 31 martie 2014. Accesat la 2 aprilie 2014.
10. „MySQL: Project Summary”. Ohloh. Black Duck Software. Accesat la 17 septembrie 2012.
11. „Supported Platforms: MySQL Database”. Oracle. Accesat la 24 martie 2014.
12. MySQL :: MySQL Downloads
13. DB-Engines Ranking
14. [The Life Cycle of a JSP Page (Sun documentation)](http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/JSPIntro4.html)  [Understanding JavaServer Pages Model 2 architecture (JavaWorld)](http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html)
15. [Forum thread (JavaRanch): OutputStream already obtained](http://www.coderanch.com/t/286297/JSP/java/OutputStream-already-obtained#2107162)
16. [Tag Libraries Tutorial - What is a Tag Library? (Sun)](http://java.sun.com/products/jsp/tutorial/TagLibraries3.html#63159)
17. http://aipi2015.andreirosucojocaru.ro/laboratoare/laborator02#clasa\_query