**MINISTERUL EDUCAŢIEI, CULTURII ȘI CERCETĂRII**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare Informatică și Microelectronică**

**Departamentul Ingineria Software și Automatică**

**Proiect de an**

**Disciplina:**Tehnici și Mecanisme de Proiectare Software

**Tema:** Implementarea și proiectarea unui sistem pentru

un magazin de componente pentru calculator

(In engleza tema)

|  |  |  |
| --- | --- | --- |
| **Student:** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Zavorot Daniel, TI-194** |
| **Coordonator:** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Cebotari Daria, asis. univ.** |
|  |  |  |

**Chișinău, 2022**

**Cuprins**

[Introducere 2](#_Toc97828737)

[1 Analiza domeniului de studiu 3](#_Toc97828738)

[1.1 Scopul, obiectivele și cerințele sistemului 4](#_Toc97828739)

[1.2 Analiza sistemelor deja existente 5](#_Toc97828740)

[2. Realizarea sistemului 7](#_Toc97828741)

[2.1 Proiectarea aplicației 9](#_Toc97828742)

[2.2 Descrierea tehnologiilor pentru sistem 13](#_Toc97828743)

[2.3 Descrierea la nivel de cod pe module 13](#_Toc97828744)

[3. Documentarea produsul realizat 18](#_Toc97828745)

[Concluzii 19](#_Toc97828746)

[Bibliografie 20](#_Toc97828747)

[Anexa A 21](#_Toc97828748)

# Introducere

Cumpărăturile sunt o activitate în care un client răsfoiește bunurile sau serviciile disponibile prezentate de unul sau mai mulți retaileri cu potențiala intenție de a cumpăra o selecție adecvată a acestora. O tipologie a tipurilor de cumpărători a fost dezvoltată de cercetători care identifică un grup de cumpărători ca cumpărători recreativi, adică cei cărora le place cumpărăturile și le privesc ca pe o activitate de agrement. Cumpărăturile online au devenit un perturbator major în industria comerțului cu amănuntul, deoarece consumatorii pot acum să caute informații despre produse și să plaseze comenzi de produse în diferite regiuni. Comercianții cu amănuntul online își livrează produsele direct acasă, la birouri sau oriunde doresc consumatorilor. Procesul B2C (business to consumer) a făcut mai ușor pentru consumatori să selecteze orice produs online de pe site-ul web al unui retailer și să fie livrat relativ rapid. Folosind metode de cumpărături online, consumatorii nu trebuie să consume energie vizitând fizic magazinele fizice. Astfel economisesc timp și costul călătoriei. Un exemplu de astfel magazin este un magazine de componente pentru calculator care in ultimii ani a ajuns in top in ramurile economice a societății.

Lucrarea dată urmărește scopul de a elabora crearea unei aplicații simple care stă la baza unui magazin online ce conține componente pentru calculator.

Lucrarea este structurată în trei capitole, unde se va analiza concret modul de proiectare și de realizare a proiectului dat, tehnologiile utilizate dar si modul de realizare a unei astfel de aplicații.

Primul dintre aceste capitole este o introducere generală în proiect, acesta include scopul, obiectivele și cerințele acestui sistem. În același timp, acest capitol analizează aria de studiu și sistemele deja existente de acest tip (Computer Univers, Amazon etc.).

Al doilea capitol include implementarea acestui sistem, în acest capitol este implementarea la nivel de cod al aplicației și implementarea modelelor de proiectare în proces. Este necesar să se dezvolte această aplicație și să se descrie tehnologiile utilizate. Descrierea codului se va face pe module, o atenție deosebita se va acorda modelelor de design folosite (Builder, Singletone etc.).

În ultimul capitol, vorbim despre aplicarea generală și facem documentația produsului. Acest capitol conține informații despre cum funcționează produsul. Elementele documentației unei aplicații sunt: ​​numele aplicației, caracteristicile/funcționalitățile, imagini cu descrierea respectiva.

# Analiza domeniului de studiu

Aplicația pentru magazinul online cu componente pentru calculator va fi o aplicație de tip GUI scrisă in limbajul Java. Aceasta aplicație va fi folosita ca un exemplu de aplicații pentru companiile ce dețin magazine online, însă nu au propria aplicație. Desigur la moment aplicațiile GUI pierd din popularitate, deoarece toți trec pe aplicații de tip Web, dar totuși o astfel de aplicație nu ar încurca pentru o companie de top.

În ingineria software, un model de proiectare software este o soluție generală, reutilizabilă la o problemă care apare frecvent într-un context dat în proiectarea software. Nu este un design finit care poate fi transformat direct în cod sursă sau mașină. Mai degrabă, este o descriere sau un șablon pentru cum se rezolvă o problemă care poate fi utilizată în multe situații diferite. Modelele de proiectare sunt cele mai bune practici formalizate pe care programatorul le poate folosi pentru a rezolva probleme comune atunci când proiectează o aplicație sau un sistem. Modelele de proiectare orientate pe obiecte arată în mod obișnuit relații și interacțiuni între clase sau obiecte, fără a specifica clasele sau obiectele finale ale aplicației care sunt implicate. Modelele care implică o stare mutabilă pot fi nepotrivite pentru limbaje de programare funcționale. Unele modele pot deveni inutile în limbaje care au suport încorporat pentru rezolvarea problemei pe care încearcă să o rezolve, iar modelele orientate pe obiecte nu sunt neapărat potrivite pentru limbajele neorientate pe obiecte. Modelele de proiectare pot fi privite ca o abordare structurată a programării computerelor intermediare între nivelurile unei paradigme de programare și un algoritm concret.[1]

In Figura 1 – Sunt arătate 3 tipuri de categorii pentru design pattern-uri:

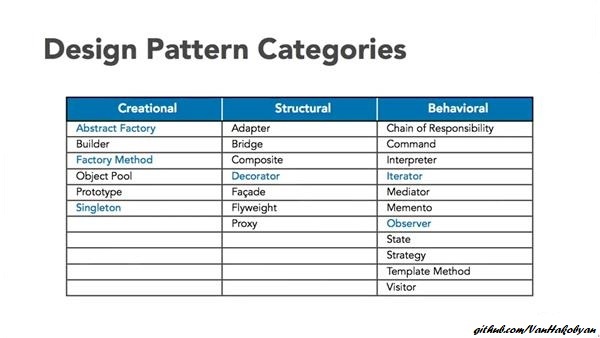


Figura 1 **-** Categorizarea design pattern-urilor

# 1.1 Scopul, obiectivele și cerințele sistemului

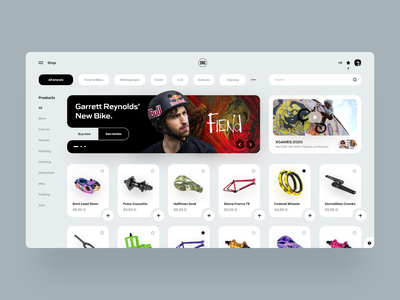
Scopul acestui proiect este crearea unui exemplu de aplicație de tip GUI pentru companiile de top ce vând produse în mediul online. Aplicația va conține un meniu ce va conține componente pentru calculator împărțite în diferite categorii. Design-ul aplicațiile (culorile) vor fi de timp închis implicit, dar cu posibilitatea schimbării în culori deschise cu ajutorul unui switch care va fi în meniul aplicație. De asemenea aplicația va conține o bază de date unde se va conține informațiile despre fiecare componentă. Nu în ultimul rând, baza de date va conține imagini pentru fiecare componentă care sunt localizate local într-un folder „images” sau pe un cloud (de exemplu Imgur).

Obiectivele de baza sunt:

* Realizarea unei aplicații de tip magazin online;
* Utilizarea șabloanelor de proiectare;
* Utilizarea minim a 3 șabloane de proiectare;
* Respectarea principiilor SOLID.

Cerințele sistemului sunt:

* Sistemul de operare: Windows;
* RAM: <1 GB;
* Prezenta jre-ului (poate fi descărcat de pe java.com);
* Prezența conexiunii la internet.



**Figura 2** Exemplu de aplicație desktop cu tema „shopping”

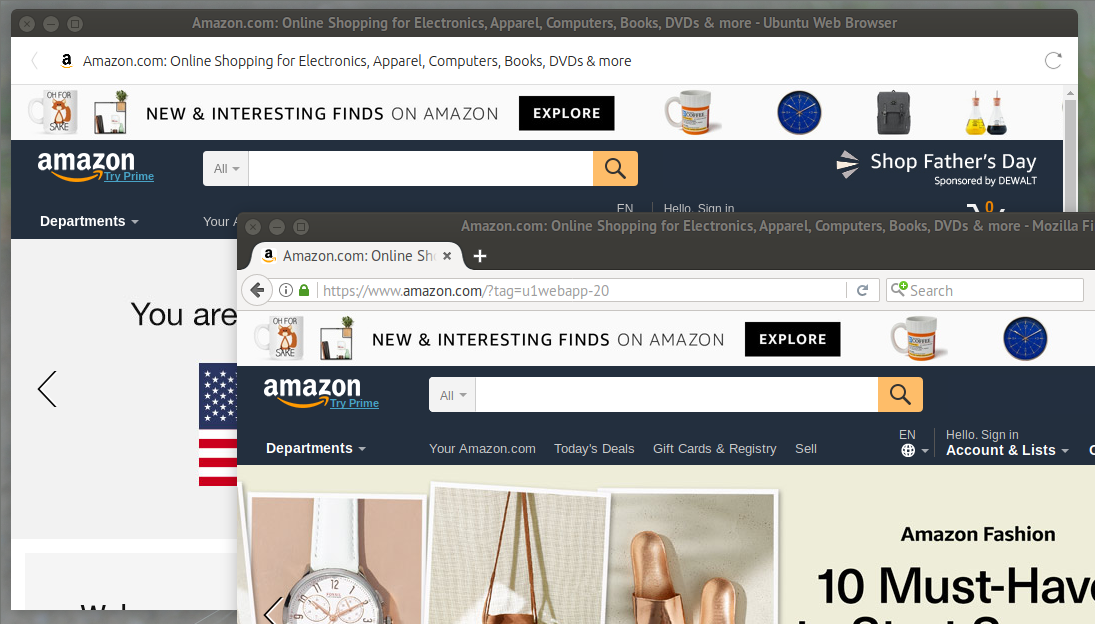
# 1.2 Analiza sistemelor deja existente

Sisteme asemănătoare cu sistemul dat sunt:

* Amazon;
* eBay;
* Computer Univers.

Amazon.com, Inc. este o companie multinațională americană de tehnologie care se concentrează pe comerțul electronic. Beneficiile aplicației companiei „Amazon” sunt:

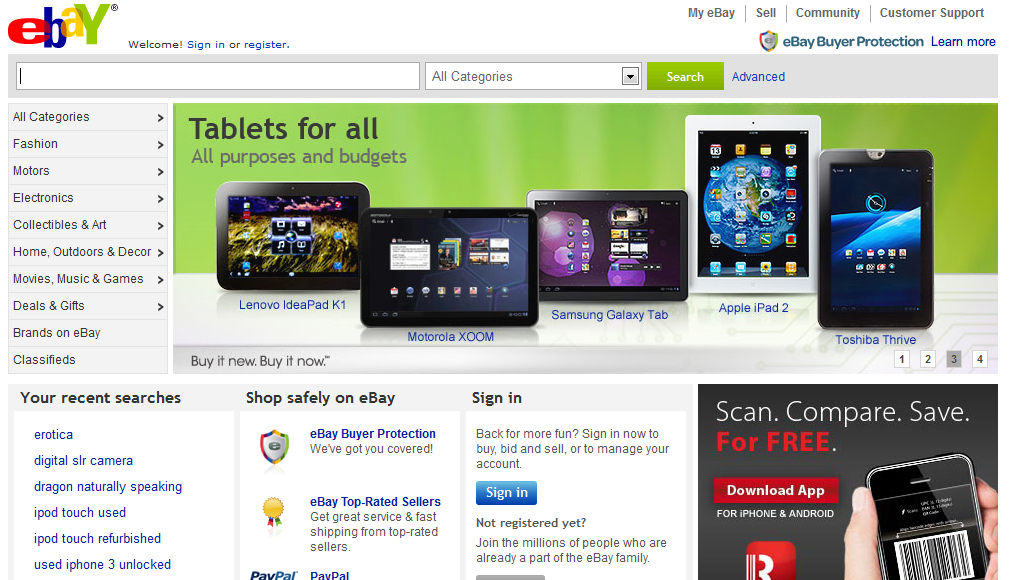
* Cumpărături cu ajutorul asistenței vocale „Alexa”
* Recomandări de produse, carduri cadou, lista de dorințe, urmărire comenzi prin localizator GPS
* Preturile produselor pot fi comparate una cu alta, de asemenea este posibilitatea cumpărării produsului cu ajutorul scanerului de coduri (QR Code).
* Trimiteți și partajați linkuri pentru produse pe orice rețea de socializare
* Opțiunile de plată online cu card de debit și credit sunt criptate în siguranță
* Notificări automate de expediere[2]



**Figura 3** Web aplicația companiei Amazon

eBay Inc. este o companie americană de comerț și shopping online care deține diverse site-uri web și face afaceri pe Internet. Avantajele companiei „eBay” sunt:

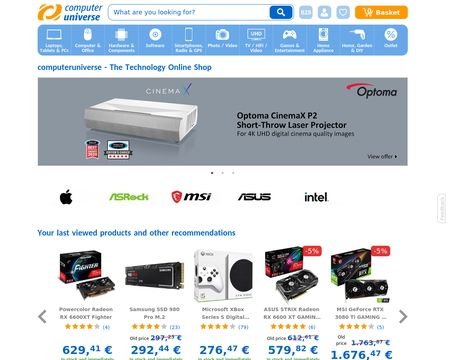
* Scanarea simplă a codurilor de bar care permite vinderea articolele cu ușurință
* Navigarea, listarea, vânzarea, se fac intuitiv în aplicație
* Numărătoare inversă în timp real pentru licitații
* Alerte de notificare pentru a actualiza utilizatorii pentru cele mai bune articole, licitații, vânzare
* Plăți online securizate cu card de debit și credit în aplicație[3]



**Figura 4** Web aplicația companiei eBay

Computeruniverse este o companie care se axează pe vânzarea componentelor pentru calculator. Avantajele acestei companii sunt:

* Staff-ul companiei sunt programatori cu experiența
* Compania este singura de așa tip cu certificat oficial
* Expedierea produsului este rapida
* 100 de produse noi in fiecare zi, iar actual sunt peste 100.000 de produse la vânzare
* 23 de ani de experiența (fondata in 1999)[4]



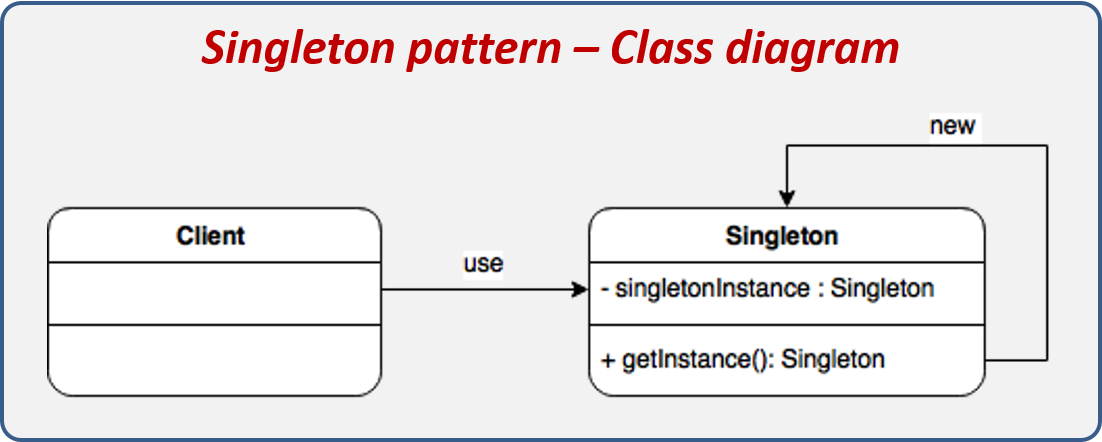
**Figura 5** Web aplicația companiei Computer Universe

# 2. Realizarea sistemului

Aplicația creata la momentul scrierii acestui capitol are implementat 4 design pattern-uri:

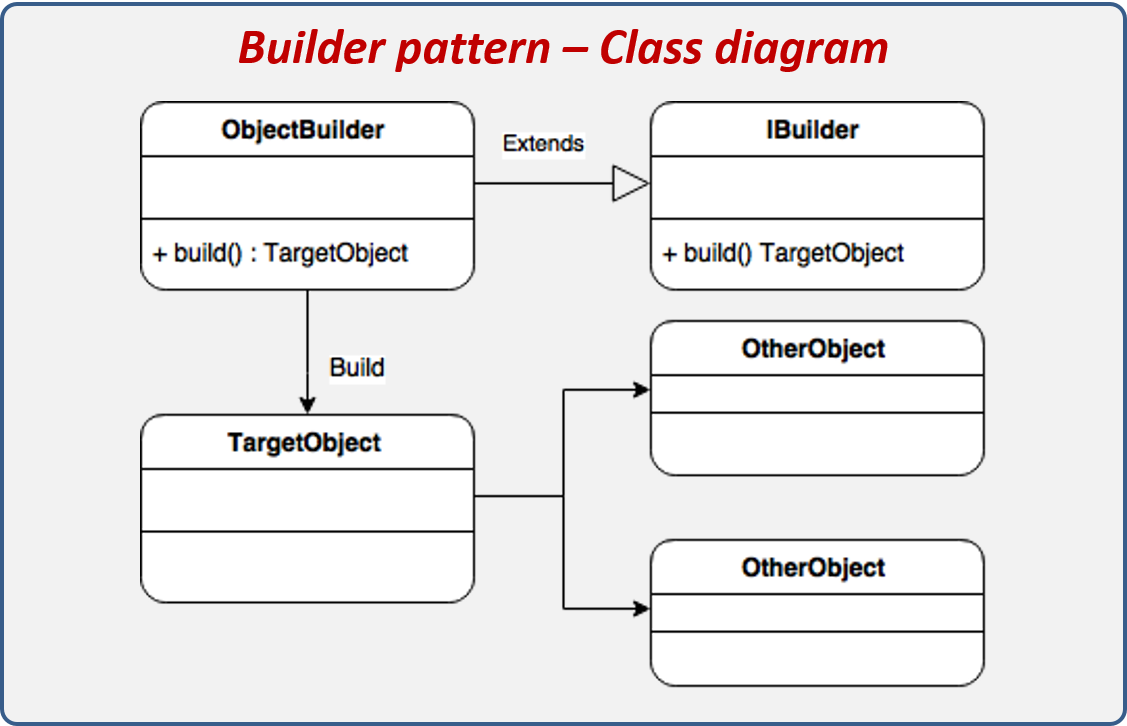
1. Singletone
2. Builder
3. Proxy
4. Observer

Primul șablon „Singletone” oferă posibilitatea rezolvării problemei pentru inițializarea aplicației și de asemenea oferă o rezolvare pentru clasa “Account” care trebuie sa aibă doar o instanță care va fi utilizată până la închiderea aplicației.



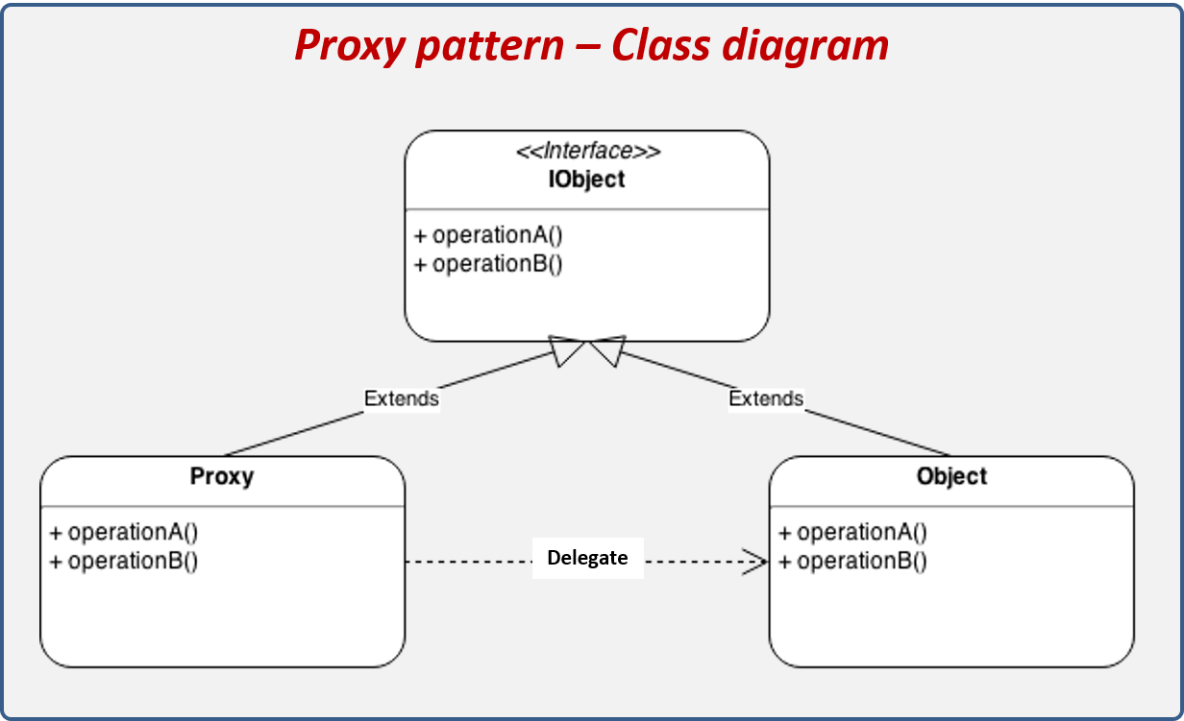
**Figura 6 –** Diagrama de clase pentru șablonul “Singletone”

Al doilea șablon “Builder” este folosit la citirea bazei de date (ca baza de date am folosit file-urile cu extensia .json, prin urmare folosesc o baza de date NoSQL) si crearea obiectelor pentru fiecare componenta care ulterior vor fi afișata pe pagina de start a aplicației si de asemenea vor fi afișate după categorii daca utilizatorul va folosi filtru pentru afișarea lor.



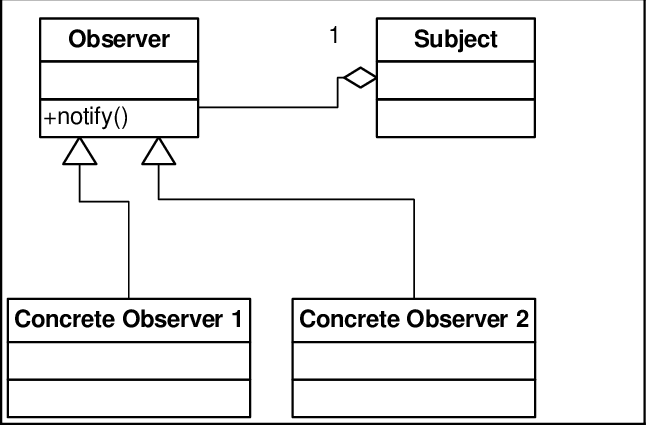
**Figura 7 –** Diagrama de clase pentru șablonul “Builder”

Al treilea șablon “Proxy” l-am folosit pentru obținerea listei cu componente, prima data lista va fi creata si citita din baza de date, apoi când va trebuie informațiile despre componente, ele vor fi luate din lista fără apelarea bazei de date, prin urmare file-urile cu extensia .json vor fi citite doar o data la pornirea aplicației, ce permite optimizarea la nivel de cod si la nivel de timp.



**Figura 8 –** Diagrama de clase pentru șablonul “Proxy”

Al patrulea șablon “Observer” este folosit la apelarea metodei de logare a utilizatorului. Când apare fereastra de logare, observer-ul se pornește si așteaptă răspuns de la utilizator, răspunsul poate fi de 2 tipuri: utilizatorul s-a logat cu succes sau utilizatorul a închis pagina de logare. Pentru prima varianta utilizatorul este redirecționat pe pagina principala, in al doilea caz fereastra de logare dispare, iar in locul ei apare funcționalul care dorește utilizatorul după apelarea unui buton, iar Thread-ul folosit pentru recepționarea răspunsului este stopat.

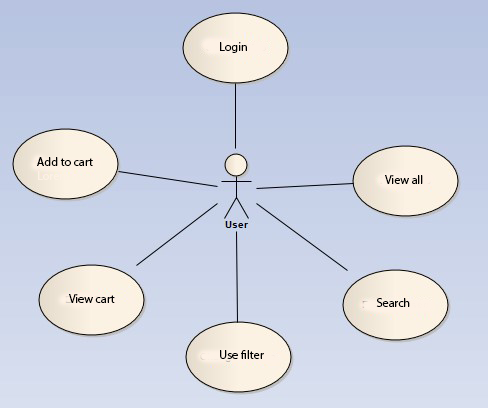


**Figura 9 –** Diagrama de clase pentru șablonul “Observer”

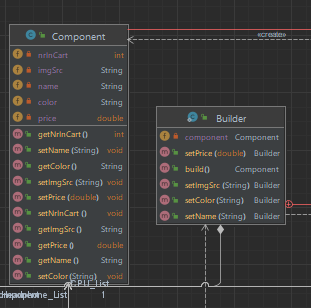
# 2.1 Proiectarea aplicației

Prin modelarea conceptuală a datelor (analiza si definirea cerințelor) se urmărește construirea unui model al datelor care să asigure transpunerea exactă a realității din domeniul analizat, fără a lua în considerare cerințele specifice unui model de organizare a datelor (cum este modelul relațional), criteriile de calitate privind organizarea datelor, cerințelor nefuncționale ale sistemului și criteriile de performanta privind stocarea și accesarea datelor.

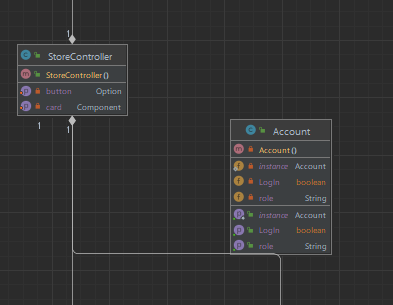
In figura 10 este reprezentată diagrama interacțiunilor utilizatorului cu sistemul dat. Utilizatorul poate interacționa cu sistemul dat prin mai multe metode.



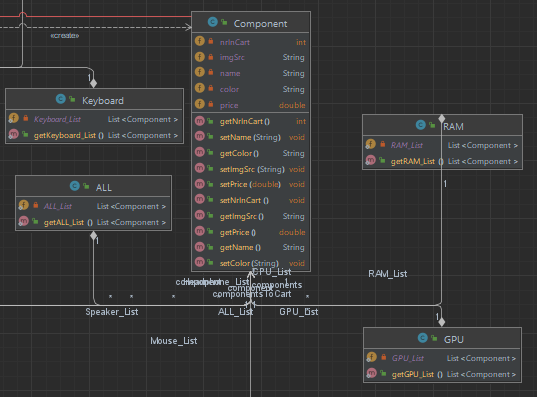
**Figura 10 –** Diagrama de interacțiune a user-ului cu sistemul



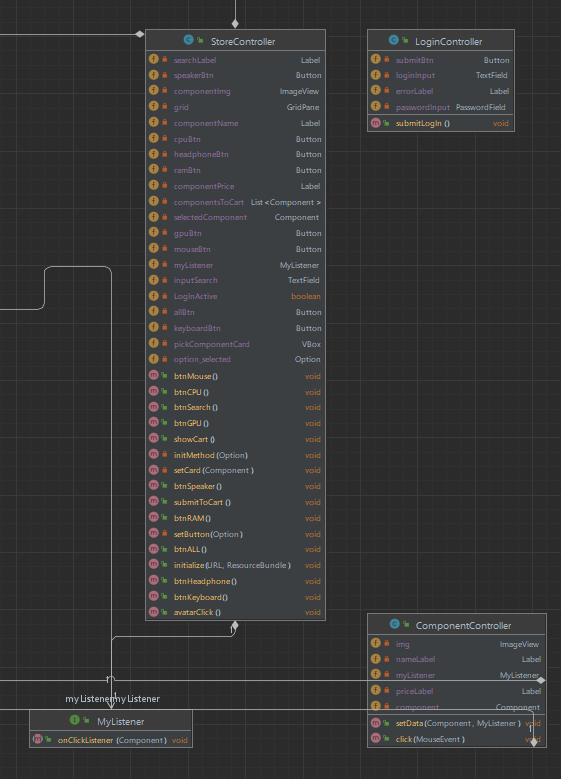
**Figura 11 –** Diagrama pentru șablonul “Builder”



**Figura 11 –** Diagrama pentru șablonul “Singletone”



**Figura 12 –** Diagrama pentru șablonul “Proxy”

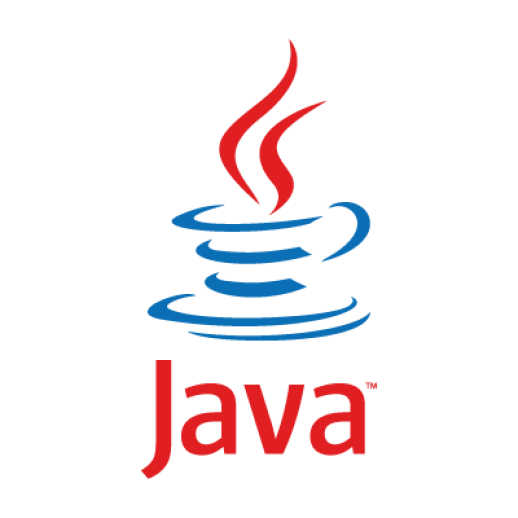


**Figura 13 –** Diagrama pentru șablonul “Observer”

# 2.2 Descrierea tehnologiilor pentru sistem

Pentru realizarea acestei aplicații a fost ales limbajul Java. Toata aplicația fiind scrisa in Java.

Java este un limbaj de programare orientat-obiect, puternic tipizat, conceput de către James Gosling la Sun Microsystems (acum filială Oracle) la începutul anilor ʼ90, fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile, spre exemplu telefon, agenda electronică, palmtop etc. În felul acesta se creează o platformă unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat în prezent cu succes și pentru programarea aplicațiilor destinate intranet-urilor. Limbajul împrumută o mare parte din sintaxă de la C și C++, dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java (engleză Java Virtual Machine, prescurtat JVM). Acest nivel de portabilitate (inexistent pentru limbaje mai vechi cum ar fi C) este posibil deoarece sursele Java sunt compilate într-un format standard numit cod de octeți (engleză byte-code) care este intermediar între codul mașină (dependent de tipul calculatorului) și codul sursă. Mașina virtuală Java este mediul în care se execută programele Java. În prezent, există mai mulți furnizori de JVM, printre care Oracle, IBM, Bea, FSF. În 2006, Sun a anunțat că face disponibilă varianta sa de JVM ca open-source.[5]



**Figura 14 –** Logo-ul companiei “Java”

Principala librărie utilizata in elaborarea acestei aplicații a fost JavaFX. JavaFX este o platformă bazată pe Java pentru construirea de aplicații GUI. Poate fi folosit atât pentru a crea aplicații desktop care rulează direct din sistemele de operare, cât și pentru aplicații de Internet ( RIA ) care rulează în browsere , cât și pentru aplicații pe dispozitive mobile[6]



**Figura 15 –** Logo-ul librăriei “JavaFX”

Pentru crearea file-ului de tip .fxml(partea vizuala) a fost utilizat Scene Builder.

Scene Builder este un instrument de proiectare interactiv GUI pentru JavaFX. Creat de Oracle, vă permite să construiți rapid interfețe cu utilizatorul fără a fi nevoie să știți cum să programați. Software-ul este disponibil în două versiuni: una (8.x) pentru JavaFX 8 și cealaltă (9.0 și +) pentru JavaFX 9 și altele.[7]

****

**Figura 16 –** Logo-ul aplicației “SceneBuilder”

# 2.3 Descrierea la nivel de cod pe module

In figura 17 se verifica daca clasa are deja o instanțiere, in caz contrar se creează:

private static Account *instance*;  
  
public static synchronized Account getInstance() {  
 if (*instance* == null) {  
 *instance* = new Account ();  
 }  
 return *instance*;  
}

**Figura 17 –** Codul pentru șablonul “Singletone”

In figura 18 este reprezentat “Builder”-ul pentru clasa “Component”:

public static class Builder {  
 private Component component;  
 public Builder(){  
 component = new Component();  
 }  
  
 public Builder setImgSrc(String imgSrc) {  
 component.setImgSrc(imgSrc);  
 return this;  
 }  
  
 public Builder setName(String name) {  
 component.setName(name);  
 return this;  
 }  
  
 public Builder setPrice(double price) {  
 component.setPrice(price);  
 return this;  
 }  
  
 public Builder setColor(String color) {  
 component.setColor(color);  
 return this;  
 }  
  
 public Component build(){  
 return component;  
 }  
}

**Figura 18 –** Codul pentru șablonul “Builder”

In figura 19.1 este reprezentat clasa de baza care folosește șablonul “Proxy”, daca lista este goala atunci se apelează metoda “readJSON”, adică se citește baza de date, in caz contrar se întoarce lista deja încărcata:

public class ALL {  
 private static List<Component> *ALL\_List* = new ArrayList<>();  
  
 public static List<Component> getALL\_List() {  
 if(*ALL\_List*.size() > 0){  
 return *ALL\_List*;  
 } else {  
 *ALL\_List*.addAll(JSON.*readJSON*(Option.*ALL*));  
 return *ALL\_List*;  
 }  
 }  
}

**Figura 19.1 –** Codul pentru șablonul “Proxy”

In figura 19.2 este reprezentat clasa “JSON” cu metoda “readJSON” care este apelata la prima chemare a listei unui component. Metoda data apelează baza de date si încarcă in lista propriu-zisa componentele din file-ul cu extensia .json:

public class JSON {  
 public static List<Component> readJSON(Option option){  
 List<Component> components = new ArrayList<>();  
 String fileName = null;  
 boolean all = false;  
 switch (option){  
 case *GPU* -> fileName = "gpu.json";  
 case *CPU* -> fileName = "cpu.json";  
 case *HEADPHONE* -> fileName = "headphone.json";  
 case *SPEAKER* -> fileName = "speaker.json";  
 case *RAM* -> fileName = "ram.json";  
 case *KEYBOARD* -> fileName = "keyboard.json";  
 case *MOUSE* -> fileName = "mouse.json";  
 case *ALL* -> all = true;  
 }  
 if (!all) {  
 boolean pass = false;  
 JsonArray array = null;  
 try {  
 array = Json.*createReader*(new FileReader("src/main/resources/md/dani3lz/tmps\_project/json/" + fileName)).readArray();  
 pass = true;  
 } catch (FileNotFoundException e) {  
 System.*out*.println("File-ul nu a fost gasit!");  
 }  
  
 if (pass) {  
 Component component;  
 for (int i = 0; i < array.size(); i++) {  
 JsonObject object = array.getJsonObject(i);  
 component = new Component.Builder()  
 .setName(object.getString("name"))  
 .setPrice(object.getInt("price"))  
 .setColor(object.getString("color"))  
 .setImgSrc(object.getString("imgSrc"))  
 .build();  
 components.add(component);  
 }  
 }  
 } else {  
 components.addAll(GPU.*getGPU\_List*());  
 components.addAll(CPU.*getCPU\_List*());  
 components.addAll(RAM.*getRAM\_List*());  
 components.addAll(Keyboard.*getKeyboard\_List*());  
 components.addAll(Mouse.*getMouse\_List*());  
 components.addAll(Headphone.*getHeadphone\_List*());  
 components.addAll(Speaker.*getSpeaker\_List*());  
 }  
 return components;  
 }  
  
}

**Figura 19.2 –** Codul pentru șablonul “Proxy”

In figura 20.1 este reprezentata clasa “Account” cu metoda “LogIn” care permite logarea unui utilizator in sistema. Daca logarea este reușita atunci metoda întoarce o variabila boolean cu valoarea true, in caz contrat false:

public boolean LogIn(String login, String password){  
 if(!this.LogIn) {  
 boolean pass = false;  
 JsonArray array = null;  
 try {  
 array = Json.*createReader*(new FileReader("src/main/resources/md/dani3lz/tmps\_project/account/accounts.json")).readArray();  
 pass = true;  
 } catch (FileNotFoundException e) {  
 System.*out*.println("File-ul nu a fost gasit!");  
 }  
  
 if (pass) {  
 for (int i = 0; i < array.size(); i++) {  
 JsonObject object = array.getJsonObject(i);  
 if(Objects.*equals*(object.getString("login"), login) && Objects.*equals*(object.getString("password"), password)){  
 this.login = login;  
 this.password = password;  
 this.role = object.getString("role");  
 this.LogIn = true;  
 break;  
 }  
 }  
 }  
 }  
 return this.LogIn;  
}

**Figura 20.1–** Codul pentru șablonul “Observer”

In figura 20.2 este reprezentata controller-ul principal a sistemului cu metoda “avatarClick”. Aceasta metoda apelează fereastra “LogIn” si creează un Thread pentru citirea variabilei boolean. Când variabila este true atunci utilizatorul este redirecționat pe pagina principala. De asemenea citeste o alta variabila care răspunde de acțiunile utilizatorului, adică daca utilizatorul a închis fereastra de logare, atunci Thread-ul este stopat, deoarece variabila pentru logare mereu va fi false si apoi este redirecționat la pagina dorita de utilizator:

public void avatarClick() {  
 if(!Account.*getInstance*().isLogIn() && !LogInActive) {  
 pickComponentCard.setVisible(false);  
 LogInActive = true;  
 option\_selected = null;  
 setButton(Option.*SEARCH*);  
 grid.getChildren().clear();  
  
 int column = 0;  
 int row = 1;  
  
 try {  
 FXMLLoader fxmlLoader = new FXMLLoader();  
 fxmlLoader.setLocation(getClass().getResource("login.fxml"));  
 AnchorPane anchorPane = fxmlLoader.load();  
  
 grid.add(anchorPane, column++, row);  
  
 GridPane.*setMargin*(anchorPane, new Insets(10));  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 new Thread(new Runnable() {  
 @Override  
 public void run() {  
 while (true){  
 // wait for responding  
 if(Account.*getInstance*().isLogIn() || !LogInActive){  
 break;  
 }  
 }  
 Platform.*runLater*(new Runnable() {  
 @Override  
 public void run() {  
 initMethod(Option.*ALL*);  
 }  
 });  
  
 }  
 }).start();  
 }  
}

**Figura 20.2–** Codul pentru șablonul “Observer”

# 3. Documentarea produsul realizat

Nu poate fi mai mic de o pagina A4, la fel si un subcapitol... Daca nu aveți cu ce completa(deși poeți pune imagini/figuri/ tabele/scheme si deja e un capitol )

# Concluzii

Nu poate fi mai mic de o pagina A4!!!

O pagina cu concluzii ce ați făcut voi in lucrare si ce obstacole ați întimpinat... Pentru ce sete necesar dezvoltarea si utilizarea șabloanelor

# Bibliografie

1. Software Design Patterns; [Resursă electronică.] – Regim de acces:  
   <https://en.wikipedia.org/wiki/Software_design_pattern>
2. Amazon; [Resursă electronică.] – Regim de acces:  
   <https://www.spaceo.ca/blog/best-online-shopping-apps/>
3. eBay; [Resursă electronică.] – Regim de acces:  
   <https://www.spaceo.ca/blog/best-online-shopping-apps/>
4. Computer Universe; [Resursă electronică.] – Regim de acces:  
   <https://www.computeruniverse.net/en/page/more-service-with-computeruniverse>
5. Java [Resursa electronică.] – Regim de acces:

<https://ro.wikipedia.org/wiki/Java_(limbaj_de_programare)>

1. JavaFX [Resursa electronică.] – Regim de acces:

<https://ro.frwiki.wiki/wiki/JavaFX>

1. SceneBuilder [Resursa electronică.] – Regim de acces:

<https://fr.wikipedia.org/wiki/Scene_Builder>

# Anexa A

Aici poate fi codul in întregime sau tabele/figuri mari

