MINISTERUL EDUCAŢIEI, CULTURII ŞI 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
(Implementing and designing a system for a computer parts store)

Student: _____ **Zavorot Daniel, TI-194**

Coordonator: _____ **Cebotari Daria, asis. univ.**

Chișinău, 2022

Cuprins

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 respectivă.

1 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 – „Categorizarea design pattern-urilor” - sunt arătate 3 tipuri de categorii pentru design pattern-uri:

Design Pattern Categories

Creational	Structural	Behavioral
Abstract Factory	Adapter	Chain of Responsibility
Builder	Bridge	Command
Factory Method	Composite	Interpreter
Object Pool	Decorator	Iterator
Prototype	Façade	Mediator
Singleton	Flyweight	Memento
	Proxy	Observer
		State
		Strategy
		Template Method
		Visitor

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.

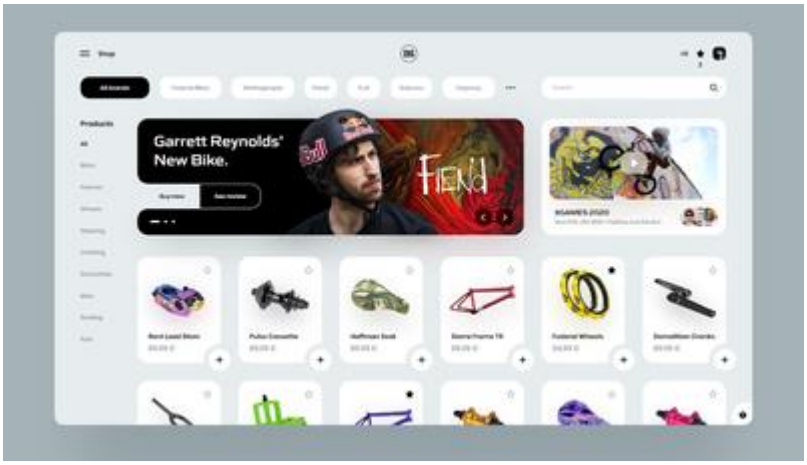

Figura 2 Exemplu de aplicație desktop cu tema „shopping”

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].

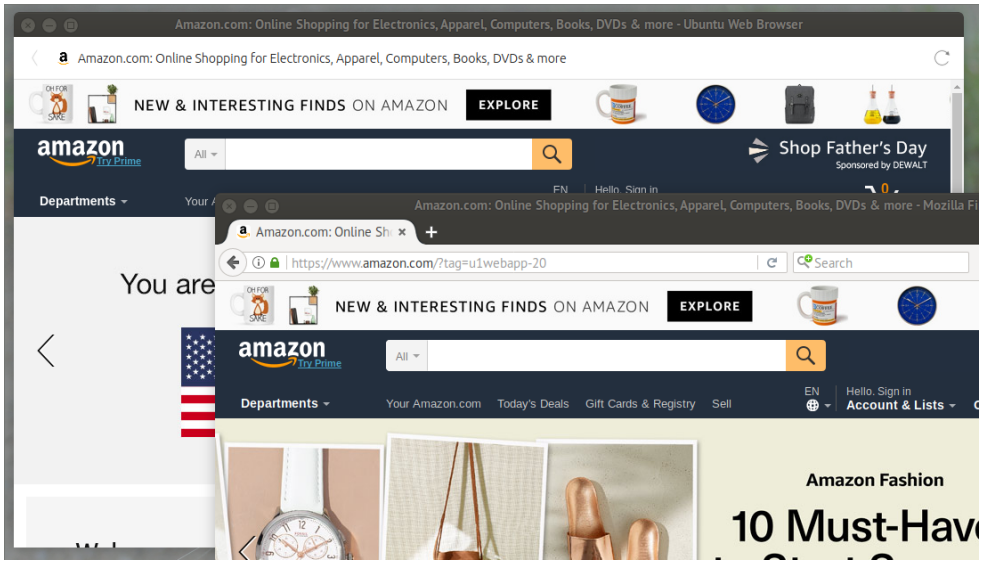

Figura 3 Web aplicația companiei Amazon

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].

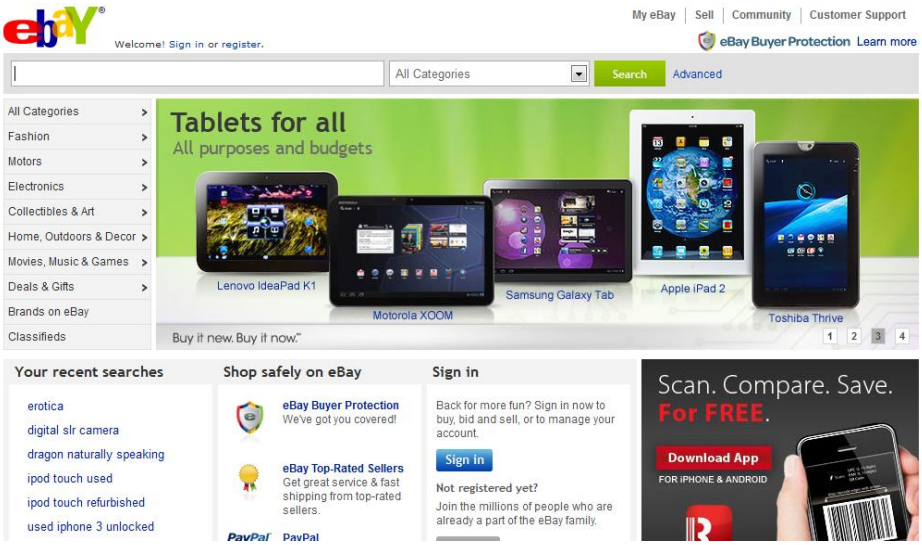

Figura 4 Web aplicația companiei eBay

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].

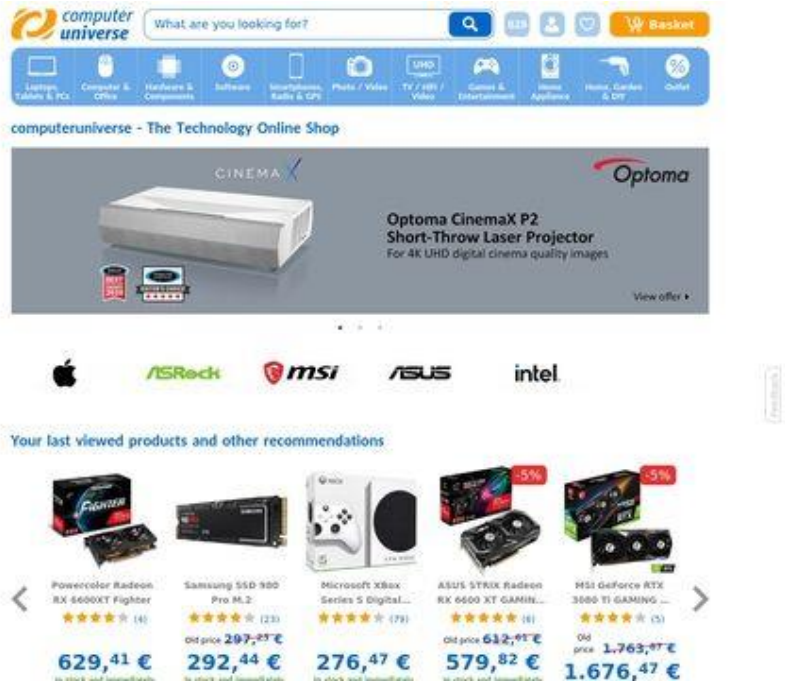

Figura 5 Web aplicația companiei Computer Universe

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.

In figura 6 este reprezentat diagrama șablonului „Singletone”:

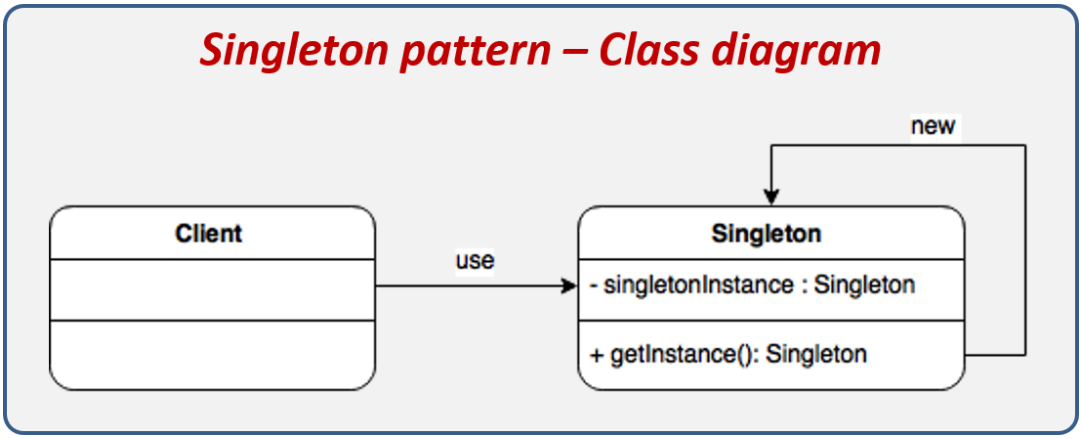

Figura 6 – Diagrama de clase pentru șablonul “Singletone”

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.

In figura 7 este reprezentat diagrama șablonului „Builder”:

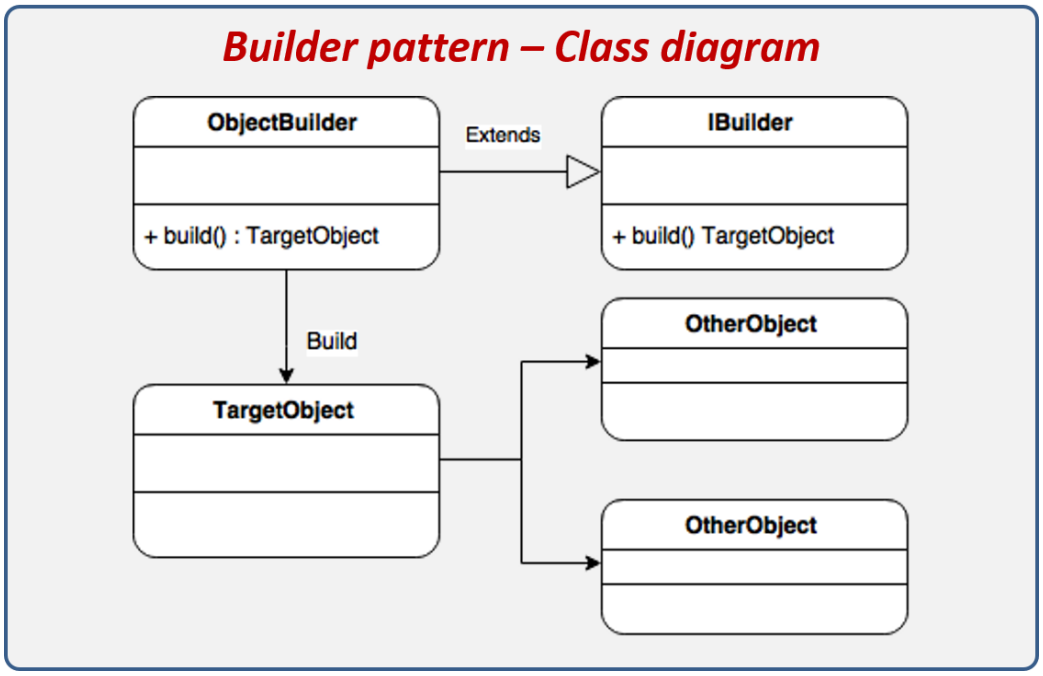

Figura 7 – Diagrama de clase pentru șablonul "Builder"

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.

In figura 8 este reprezentat diagrama șablonului „Proxy":

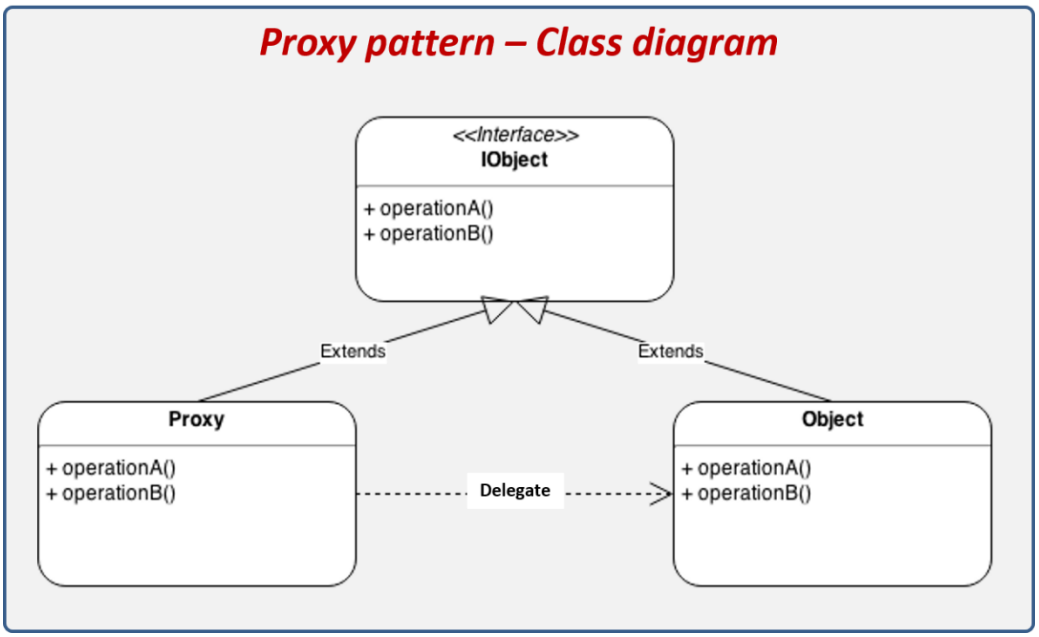

Figura 8 – Diagrama de clase pentru șablonul "Proxy"

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.

In figura 9 este reprezentat diagrama șablonului „Observer”:

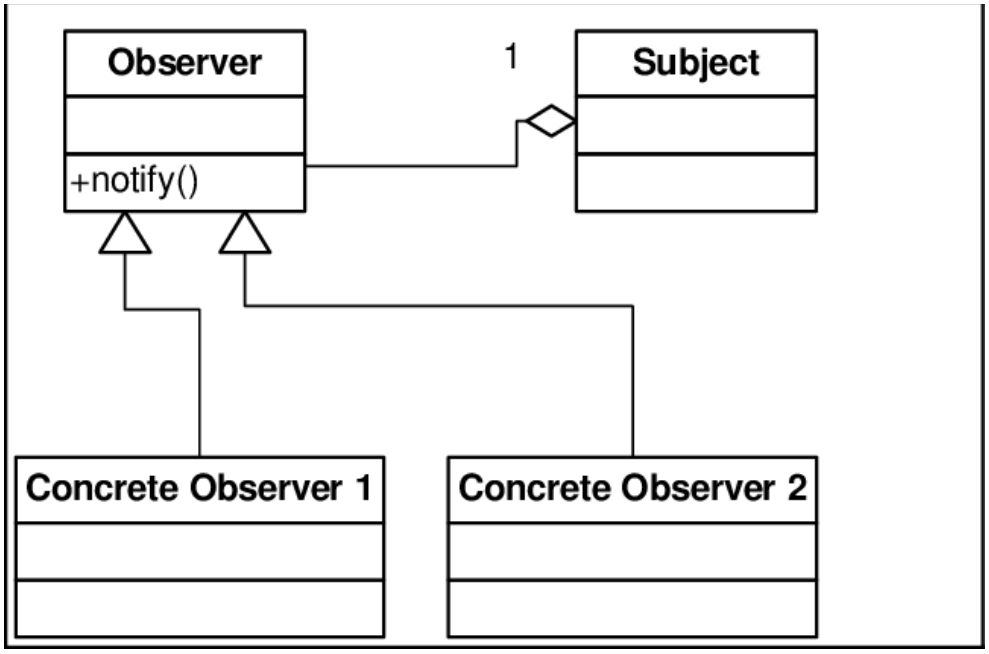

Figura 9 – Diagrama de clase pentru șablonul “Observer”

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.

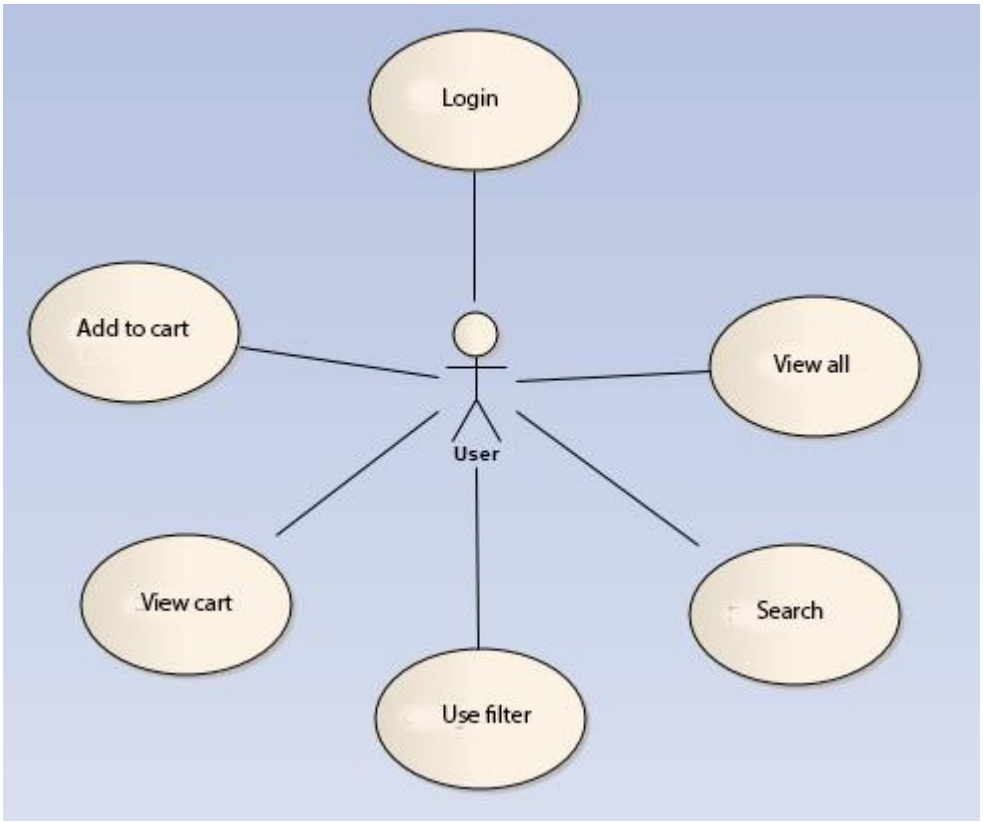

Figura 10 – Diagrama de interacțiune a user-ului cu sistemul

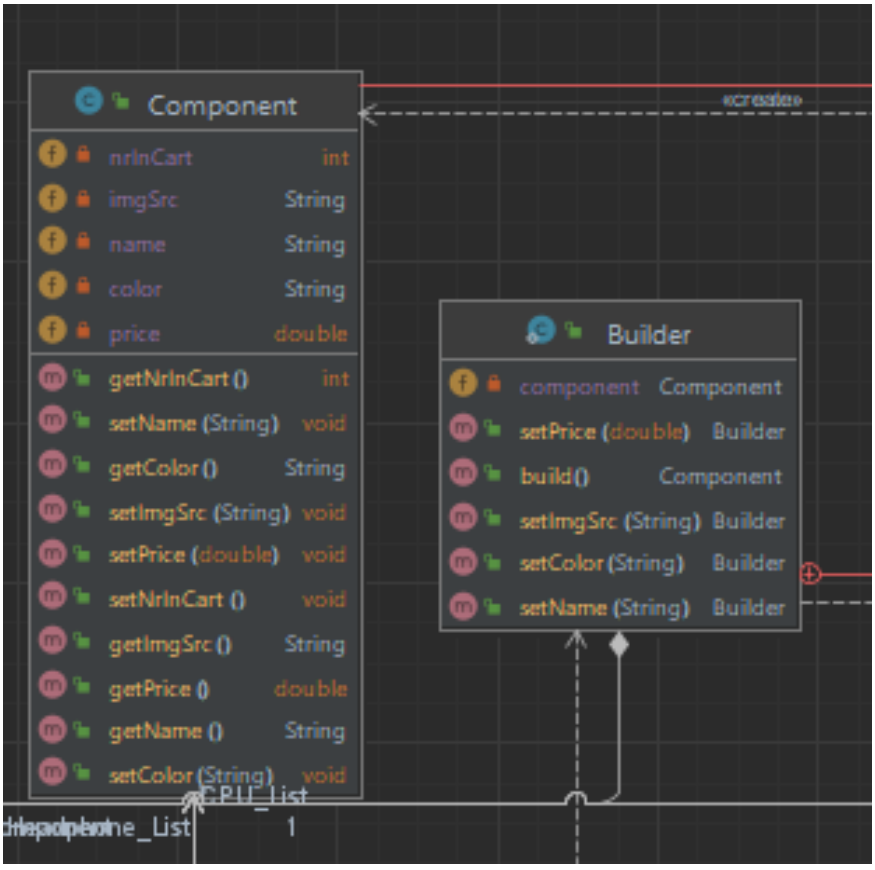

Figura 11 – Diagrama pentru șablonul "Builder"

In figura 11 este afișata diagrama șablonului „Builder” ce este implementata direct in clasa „Component” care permite crearea componentelor pentru calculator care ulterior vor fi afișate pe pagina de vânzare a aplicației.

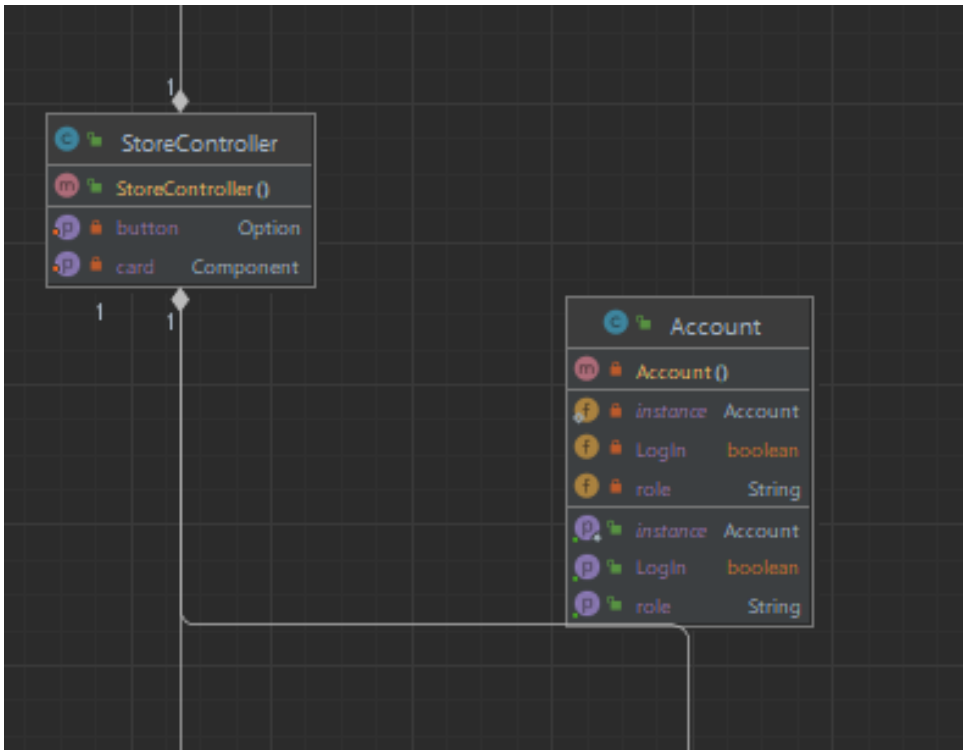

Figura 12 – Diagrama pentru șablonul “Singletone”

In figura 12 este afișata diagrama șablonului „Singletone” ce este implementata direct in clasa „Account” care permite crearea unei singure instanțe când utilizatorul s-a logat cu succes.

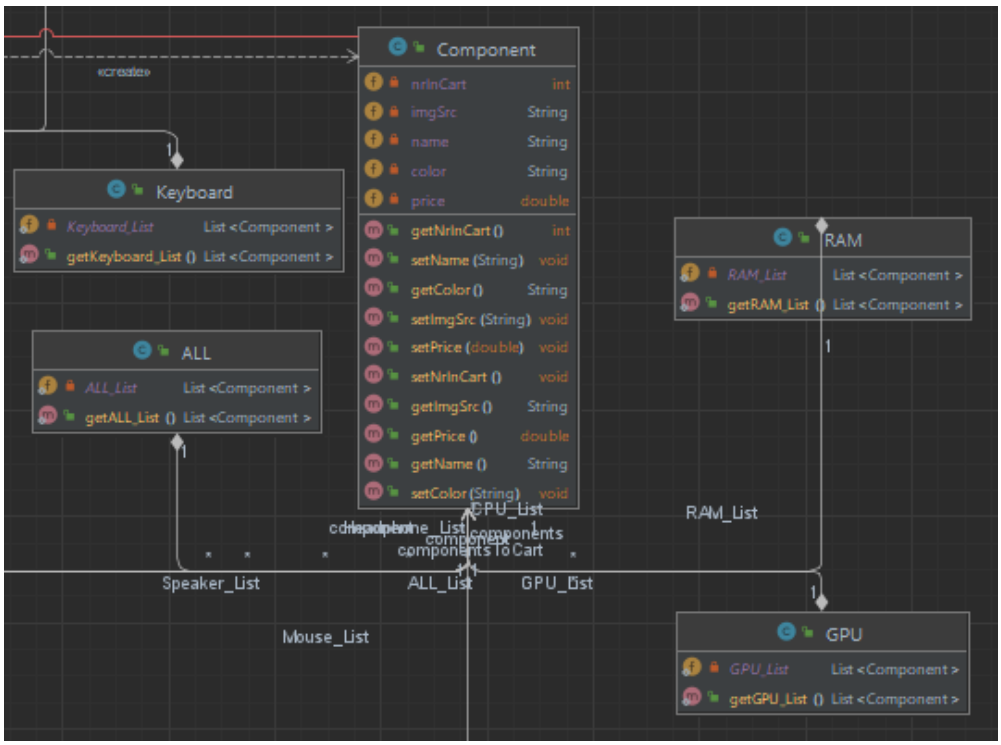

Figura 13 – Diagrama pentru șablonul “Proxy”

In figura 13 este afișata diagrama șablonului „Proxy” ce este implementata in fiecare clasa a componentelor ce exista in aplicație. Cu ajutorul acestui șablon, baza de date se încarcă doar o data in memoria aplicație si este salvata într-o lista, următoarea chemare de citire a componentelor existente va fi întoarsa aceea lista, in loc sa fie citita din nou baza de date.

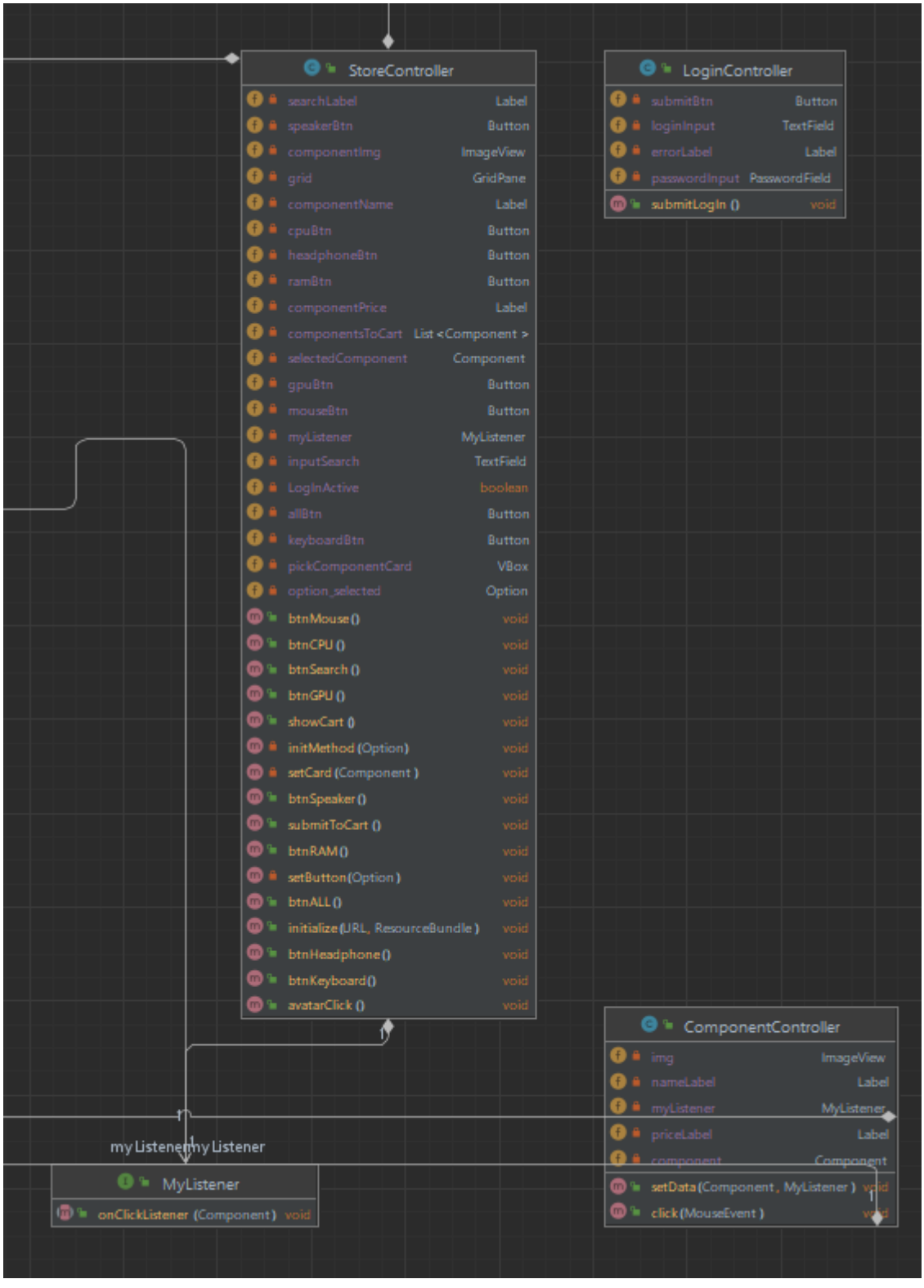

Figura 14 – Diagrama pentru șablonul “Observer”

In figura 14 este afișata diagrama șablonului „Observer” ce este implementata direct in clasa „StoreController” care permite crearea unui Thread care ulterior va aștepta un răspuns de la clasa „LoginController” daca utilizatorul a intrat cu succes in aplicație sau nu.

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 ’90, 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 15 – 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 16 – 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 17 – Logo-ul aplicației “SceneBuilder”

2.3 Descrierea la nivel de cod pe module

In următorul cod este implementat șablonul „Singletone” pentru crearea unei singure instanțe pentru clasa „Accout” ce permite stocarea datelor de intrare a utilizatorului:

```
private static Account instance;

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

In următorul cod este implementat șablonul „Builder” pentru crearea componentelor care ulterior vor fi afișate in aplicație:

```
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;
    }
}
```

In următorul cod 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;
        }
    }
}
```

In codul următor 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;
    }

}
```

In codul următor 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;
}
```

In codul următor 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();
    }
}
```

3. Documentarea produsul realizat

La pornirea aplicatiei create pentru un magazin ce contine componente pentru calculator va arata in modul urmator cum este aratat in figura 18:

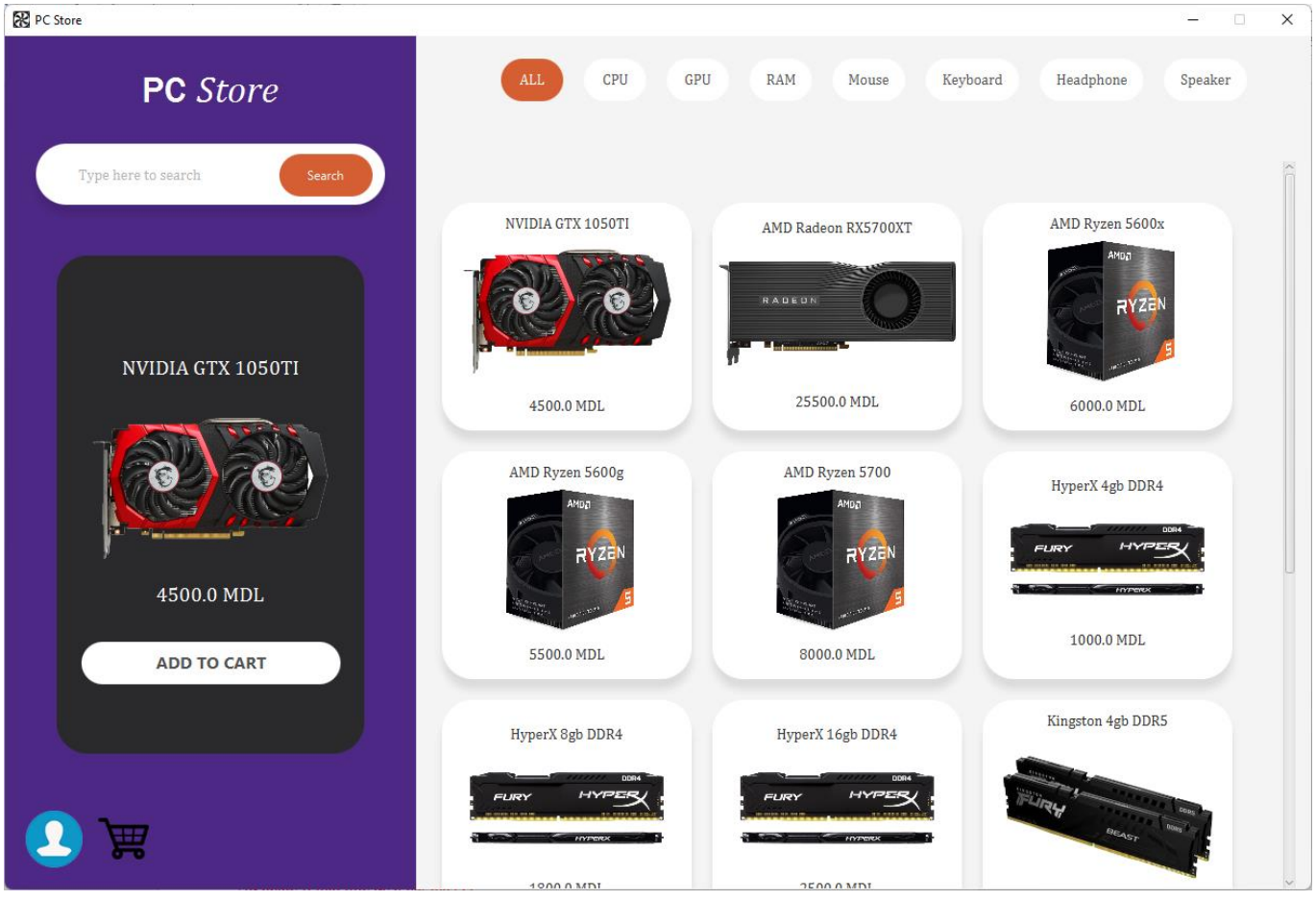

Figura 18 – Pagina de start a aplicației

Cum putem sa observam in figura 18, aplicația se deschide pe pagina cu filtru „ALL” selectat, adică se afișează absolut toate componentele disponibile pentru vânzare, de asemenea aplicația se deschide cu prima componenta selectata ce este afișata in cardboard-ul ce se afla in partea stânga a aplicației. De asemenea fiecare componenta are culoarea proprie a cardbord-ului, însă mie mi-a fost lene si am pus la toate culoarea gri. Daca ulterior vom selecta alt filtru de exemplu „RAM”, atunci vor fi afișate doar componentele care sunt clasificate in categoria aceasta și cu prima componentă selectată cum este arătat in figura 19:

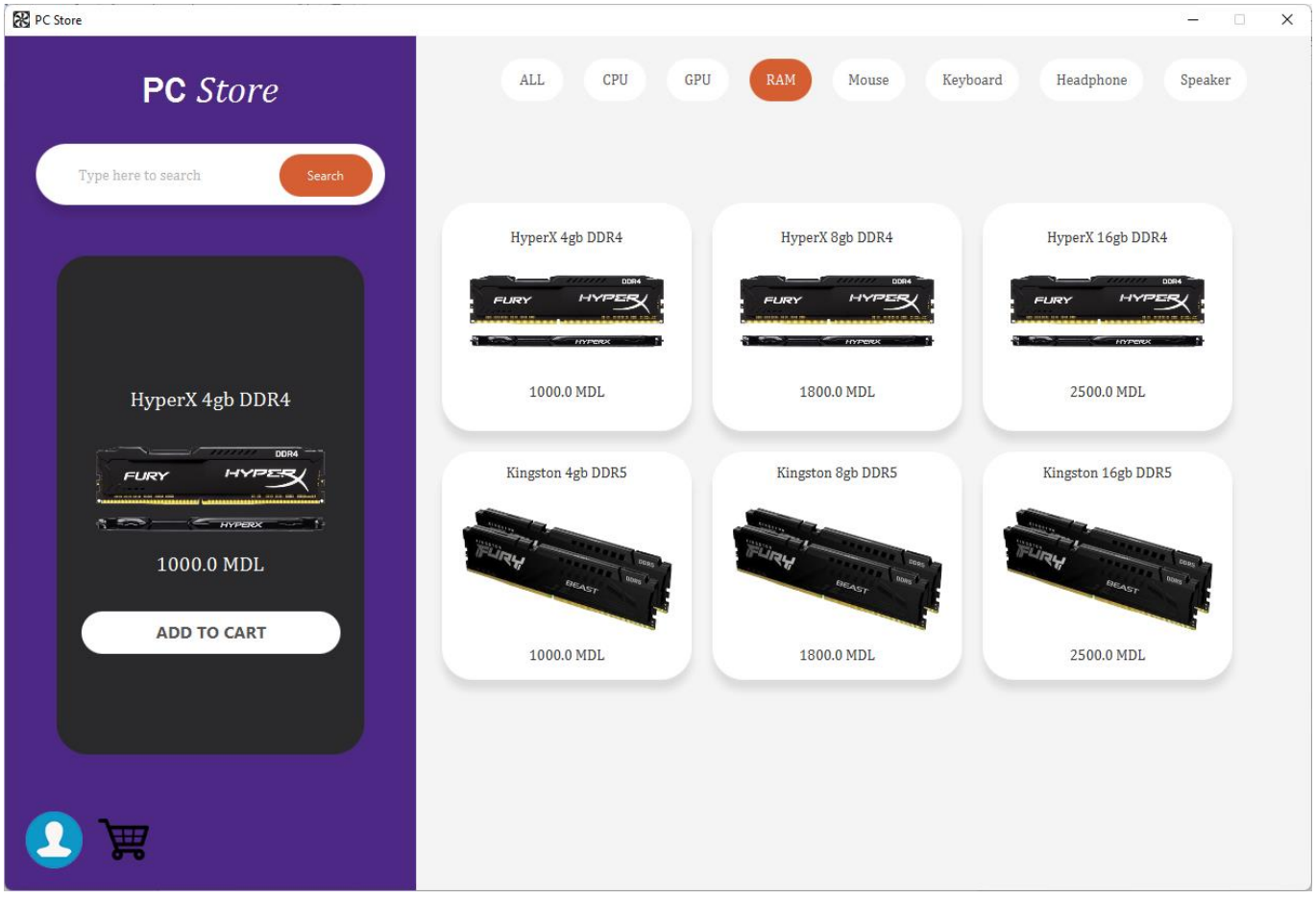

Figura 19 – Aplicația cu filtru „RAM” selectat

In figura 20 este afișata pagina de logare a aplicație poate fi accesata daca apăsam pe butonul albastru din stânga jos sau pe butonul „ADD TO CART” daca utilizatorul nu este deja logat. Pagina de logare arata in felul următor:

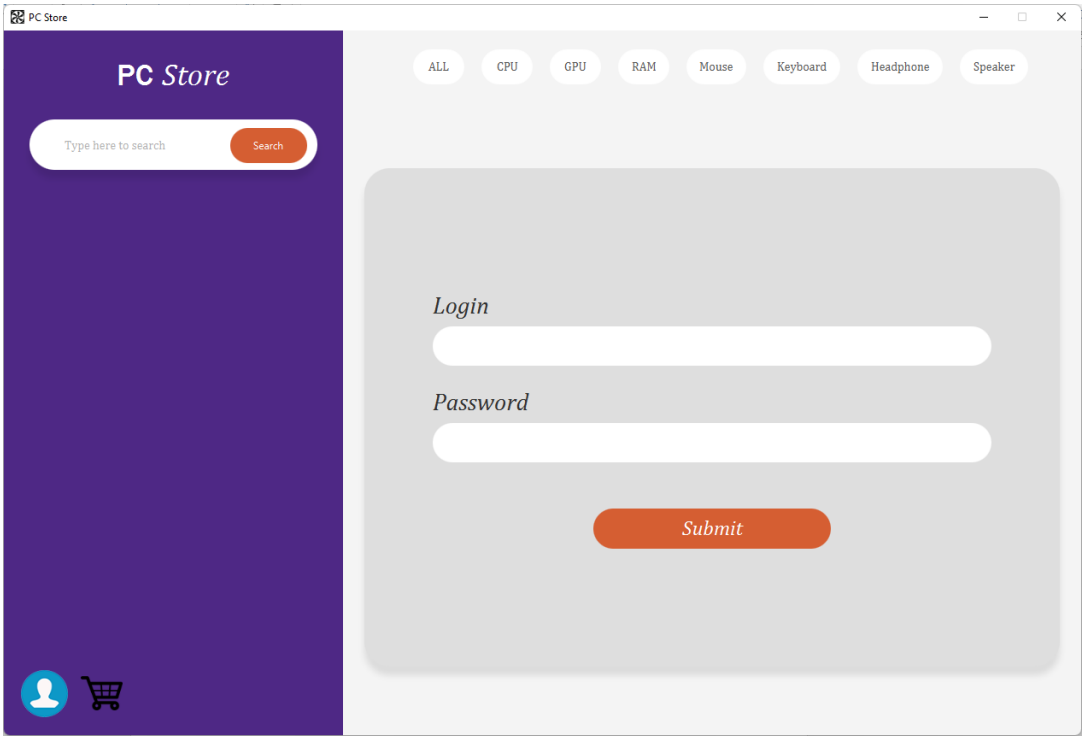

Figura 20 – Pagina de logare a aplicație

Din momentul dat este creat un Thread care așteaptă răspunsul utilizatorului (logare cu succes sau schimbarea paginii aplicației). După logarea cu succes, utilizatorul este redirecționat pe pagina principala, iar Thread-ul este stopat, din momentul acesta utilizatorul poate sa adauge componente in cart, iar butonul albastru din stânga jos devine indisponibil.

In figura 21 este afișata pagina unde sunt stocate toate componentele care au fost adăugate in cart:

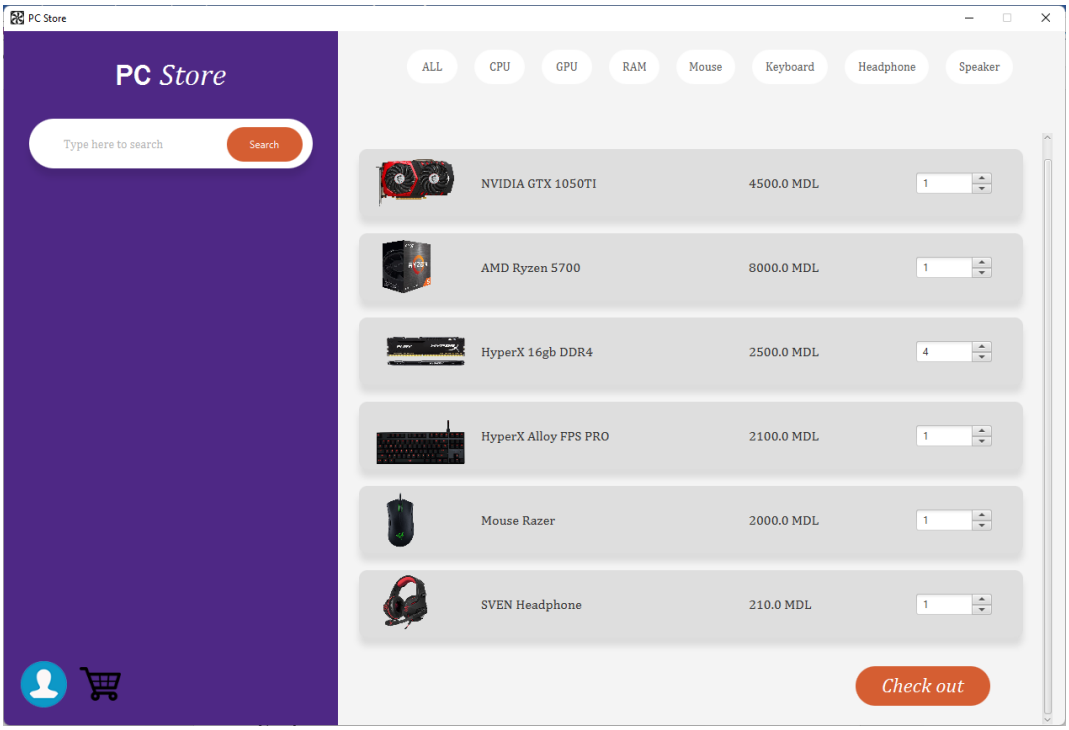

Figura 21 – CART-ul care afișează toate componentele adăugate.

Cum putem sa vedem, o componente ocupa un loc, adică cantitatea lor este afișata in dreapta fiecărui bloc, nu va fi ca aceia componente sa posede 2 sau mai multe blocuri pe această pagină. Pe aceasta pagina putem ajunge daca apăsam butonul cu simbolul „CART” din dreapta jos a aplicație.

In figura 22 putem vedea cum lucrează funcția de căutare a aplicație:

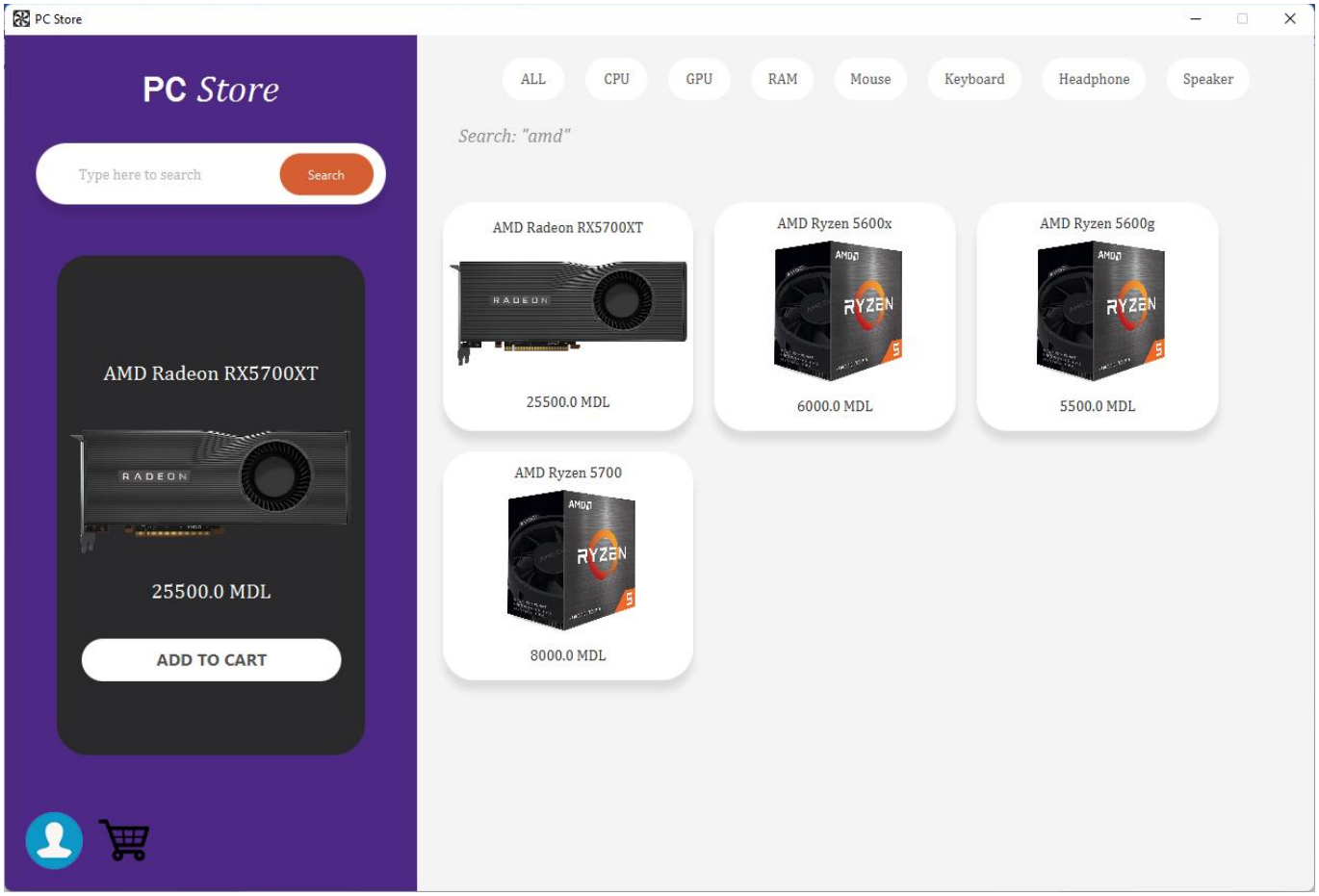

Figura 22 – Funcția de căutare componentelor

In figura 22 este reprezentat pagina afișata după căutarea cuvântului „amd”. Sub filtrele aplicației apare un label cu cuvântul căutat, iar in textfield-ul blocului de căutare dispare cuvântul căutat. Nu in ultimul rând pe pagina de magazin apar componentele ce conțin cuvântul cheie căutat cu prima componenta selectat.

Concluzii

In concluzie, după studierea bibliotecii JavaFX pentru limbajul de programare Java si crearea aplicației pentru proiectul de an am căpătat experiența ce sper ca va fi de folos in viitor. De asemenea m-am familiarizat cu aplicația „SceneBuilder” ce permite creare design-ul aplicației în real time pentru biblioteca JavaFX.

Avantajele bibliotecii JavaFX din opinia proprie este destul de ușor de lucrat cu metodele bibliotecii. De asemenea este o biblioteca destul de complexa ce permite crearea unei aplicații destul de avansate. Un alt plus a aplicației ar fi ca jumate din logica programului este stocată într-un file cu extensia .fxml care este creat de „SceneBuilder”, de asemenea putem să-l creăm de sine stătător și să facem același design, deoarece totul este structurat pe înțeles, iar logica nu este grea, însă va lua mai mult timp în comparație dacă lucrăm cu SceneBuilder-ul.

Dezavantajele întâlnite de mine sunt in primul rând cu totul este limitat, logica programului poate fi scrisa doar într-o clasa Controller ce-i aparține un file cu extensia .fxml, adică un file – o clasa si nu mai mult. Un alt dezavantaj ar fi ca biblioteca nu are o metoda pentru crearea unei metode personale de tip timer, care va fi chemate la fiecare X secunde care vom indica la crearea timer-ului. Poate și că există o astfel de metodă, dar personal de mine, după câteva ore de căutare prin documentație și prin alte site-uri nu a fost găsita. Singura ieșire din situație care am găsit-o este crearea unui Thread care va apela mereu acea metoda, însă nu putem indica o data la cat timp sa o apeleze, toata logica trebuie sa o implementam singuri.

Un minus considerabil ar mai fi ca pentru proiectul acesta trebuie de implementat șabloane care se folosesc doar in anumite situații, de asemenea JavaFX are deja implementate câteva șabloane, de exemplu Factory Method, ce face imposibil rescrierii lui. Prin urmare am implementat doar 4 șabloane: Singletone, Builder, Observer si Proxy care sunt plus-minus utile, dar in realitate era mult mai ușor fără aceste șabloane de creat aplicația pentru desktop.

In concluzie, dacă luăm în calcul avantajele și dezavantajele bibliotecii „JavaFX” a limbajului de programare „Java” este o platformă destul de bogată în metode și ușor de folosit după citirea documentației pe site-ul oficial ce nu ia mult timp, iar ca rezultat, in urma studierii acestei platforme, putem sa creăm o aplicației de tip desktop cu un GUI plus-minus normal cu un funcțional de bază.

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)
6. JavaFX [Resursa electronică.] – Regim de acces: https://ro.frwiki.wiki/wiki/JavaFX
7. SceneBuilder [Resursa electronică.] – Regim de acces: https://fr.wikipedia.org/wiki/Scene_Builder

Anexa A

```
package md.dani3lz.tmps_project;

import javafx.application.Platform;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.geometry.Insets;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Region;
import javafx.scene.layout.VBox;
import md.dani3lz.tmps_project.Assets.*;
import md.dani3lz.tmps_project.Assets.Account.Account;
import md.dani3lz.tmps_project.Assets.Options.Option;

import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;

public class StoreController implements Initializable {
    @FXML
    private Button allBtn;
    @FXML
    private Button cpuBtn;
    @FXML
    private Button gpuBtn;
    @FXML
    private Button headphoneBtn;
    @FXML
    private Button keyboardBtn;
    @FXML
    private Button mouseBtn;
    @FXML
    private Button ramBtn;
    @FXML
    private Button speakerBtn;
    @FXML
    private ImageView componentImg;
    @FXML
    private Label componentName;
    @FXML
    private Label componentPrice;
    @FXML
    private GridPane grid;
    @FXML
    private VBox pickComponentCard;
    @FXML
    private TextField inputSearch;
    @FXML
    private Label searchLabel;

    private MyListener myListener;
    private Option option_selected = null;
```

```
private boolean LogInActive = false;
private Component selectedComponent;
private List<Component> componentsToCart = new ArrayList<>();

public void submitToCart(){
    if(Account.getInstance().isLogIn()){
        new Thread(new Runnable() {
            @Override
            public void run() {
                boolean exist = false;
                for(Component component: componentsToCart){
                    if(component.getName().equals(selectedComponent.getName())){
                        component.setNrInCart();
                        exist = true;
                        break;
                    }
                }
                if(!exist) {
                    componentsToCart.add(selectedComponent);
                }
            }
        }).start();

    } else {
        avatarClick();
    }
}

public void showCart() {
    option_selected = Option.SEARCH;
    pickComponentCard.setVisible(false);
    LogInActive = false;
    grid.getChildren().clear();
    setButton(Option.SEARCH);

    int column = 0;
    int row = 1;

    try {
        for (Component component : componentsToCart) {
            FXMLLoader fxmlLoader = new FXMLLoader();
            fxmlLoader.setLocation(getClass().getResource("cart.fxml"));
            AnchorPane anchorPane = fxmlLoader.load();

            CartController cartController = fxmlLoader.getController();
            cartController.setData(component);

            grid.add(anchorPane, column, row++);

            GridPane.setMargin(anchorPane, new Insets(10));
        }

        if(componentsToCart.size() > 0) {
            FXMLLoader fxmlLoader = new FXMLLoader();
            fxmlLoader.setLocation(getClass().getResource("checkout.fxml"));
            AnchorPane anchorPane = fxmlLoader.load();
            grid.add(anchorPane, column, row);
            GridPane.setMargin(anchorPane, new Insets(10));
        }

    } catch (IOException e) {
        e.printStackTrace();
```

```
    }

}
public void btnALL() { initMethod(Option.ALL); }

public void btnGPU(){initMethod(Option.GPU);}

public void btnCPU(){initMethod(Option.CPU);}

public void btnRAM(){initMethod(Option.RAM);}

public void btnMouse(){initMethod(Option.MOUSE); }

public void btnKeyboard(){initMethod(Option.KEYBOARD); }

public void btnHeadphone(){initMethod(Option.HEADPHONE);}

public void btnSpeaker(){initMethod(Option.SPEAKER);}

public void btnSearch() {
    String textInput = inputSearch.getText().toLowerCase();
    inputSearch.clear();
    if(!textInput.equals("")) {
        searchLabel.setVisible(true);
        searchLabel.setText("Search: \"" + textInput + "\"");
        Search.searchComponents(textInput);
        initMethod(Option.SEARCH);
    }
}

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();
    }
}

private void initMethod(Option option){
    pickComponentCard.setVisible(true);
    LogInActive = false;
    if(option != Option.SEARCH){
        searchLabel.setVisible(false);
    }
    if(option_selected != option || option == Option.SEARCH) {
        option_selected = option;
        setButton(option);
        grid.getChildren().clear();
        List<Component> components;
        components = InitComponents.getInstance().getData(option);

        if (components.size() > 0) {
            setCard(components.get(0));
            myListener = new MyListener() {
                @Override
                public void onClickListener(Component component) {
                    setCard(component);
                }
            };
        }

        int column = 0;
        int row = 1;

        try {
            for (Component component : components) {
                FXMLLoader fxmlLoader = new FXMLLoader();
                fxmlLoader.setLocation(getClass().getResource("component.fxml"));
                AnchorPane anchorPane = fxmlLoader.load();

                ComponentController componentController = fxmlLoader.getController();
                componentController.setData(component, myListener);

                if (column == 3) {
                    column = 0;
                    row++;
                }

                grid.add(anchorPane, column++, row);

                GridPane.setMargin(anchorPane, new Insets(10));
            }

            grid.setMinWidth(Region.USE_COMPUTED_SIZE);
            grid.setPrefWidth(Region.USE_COMPUTED_SIZE);
            grid.setMaxWidth(Region.USE_PREF_SIZE);
```

```
                grid.setMinHeight(Region.USE_COMPUTED_SIZE);
                grid.setPrefHeight(Region.USE_COMPUTED_SIZE);
                grid.setMinHeight(Region.USE_PREF_SIZE);

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private void setCard(Component component){
        componentName.setText(component.getName());
        componentPrice.setText(component.getPrice() + " MDL");
        Image image = new Image(getClass().getResourceAsStream(component.getImgSrc()));
        componentImg.setImage(image);
        pickComponentCard.setStyle( "-fx-background-color: #"+component.getColor()+";\n" +
                "    -fx-background-radius: 30;");
        selectedComponent = component;
    }

   private void setButton(Option option) {
       allBtn.getStyleClass().remove("btn-selected");
       cpuBtn.getStyleClass().remove("btn-selected");
       gpuBtn.getStyleClass().remove("btn-selected");
       ramBtn.getStyleClass().remove("btn-selected");
       keyboardBtn.getStyleClass().remove("btn-selected");
       mouseBtn.getStyleClass().remove("btn-selected");
       headphoneBtn.getStyleClass().remove("btn-selected");
       speakerBtn.getStyleClass().remove("btn-selected");

       switch (option) {
           case ALL -> allBtn.getStyleClass().add("btn-selected");
           case MOUSE -> mouseBtn.getStyleClass().add("btn-selected");
           case KEYBOARD -> keyboardBtn.getStyleClass().add("btn-selected");
           case HEADPHONE -> headphoneBtn.getStyleClass().add("btn-selected");
           case CPU -> cpuBtn.getStyleClass().add("btn-selected");
           case GPU -> gpuBtn.getStyleClass().add("btn-selected");
           case SPEAKER -> speakerBtn.getStyleClass().add("btn-selected");
           case RAM -> ramBtn.getStyleClass().add("btn-selected");
       }
   }

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        initMethod(Option.ALL);
    }
}
```

Github: https://github.com/dani3lz/tmps_project