TEHNICI DE PROGRAMARE FUNDAMENTALE
ASSIGNMENT 3

ORDER MANAGEMENT

DOCUMENTATIE

Kovacs Alexandru
Grupa 30223

Cuprins

1. Obiectivul temei

1.1. Obiectivul principal

Obiectivul principal al acestei teme de laborator este de a proiecta si a implementa o aplicatie care permite utilizatorului administrarea unor date despre clienti, produse si comenzi. Aceasta aplicatie ofera posibilitatea de a vizualiza date, de a adauga date noi, de a edita date deja existente si de a sterge date. De asemenea, utilizatorul poate genera o factura pentru fiecare comanda existenta care va fi salvata sub forma unui fisier PDF. Aplicatia se va folosi de o baza de date (MySQL) in vederea stocarii datelor.

1.2. Obiective secundare

Obiectivele secundare care vor fi abordate in aceasta tema sunt urmatoarele:

- Analiza problemei si identificarea cerintelor – **Capitolul 2**
- Proiectarea aplicatiei de administrare a comenzilor – **Capitolul 3**
- Implementarea aplicatiei de administrare a comenzilor – **Capitolul 4**
- Testarea aplicatiei de administrare a comenzilor – **Capitolul 5**

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

2.1. Analiza problemei

Cerintele functionale care reies din analiza enuntului temei de laborator sunt:

- Aplicatia de administrare a comenzilor ar trebui sa permita utilizatorilor sa vizualizeze date despre:
  - Clienti
  - Produse
  - Comenzi
- Aplicatia de administrare a comenzilor ar trebui sa permita utilizatorilor sa adauge noi date despre:
  - Clienti
  - Produse
  - Comenzi
- Aplicatia de administrare a comenzilor ar trebui sa permita utilizatorilor sa modifice date existente despre:
  - Clienti
  - Produse
  - Comenzi
- Aplicatia de administrare a comenzilor ar trebui sa permita utilizatorilor sa stearga date despre:
  - Clienti
  - Produse
  - Comenzi
- Aplicatia de administrare a comenzilor ar trebui sa permita utilizatorilor sa genereze facturi
- Aplicatia de administrare a comenzilor ar trebui sa notifice utilizator in cazul aparitiei unei erori

2.2. Cazuri de utilizare

Caz de utilizare: adaugarea si actualizarea datelor
Actorul principal: utilizatorul
Scenariul principal:

1. Utilizatorul navigheaza la meniul dorit utilizand interfata grafica.
2. Utilizatorul apasa butonul corespunzator operatiei care urmeaza sa fie efectuata.
3. Va aparea un formular in vederea adaugarii de date.
4. Utilizatorul completeaza formularul cu datele dorite si apasa butonul „Save Changes”.
5. Formularul va disparea si un mesaj va fi afisat utilizatorului.
6. Noile modificar i vor fi acum reflectate de interfata grafica.

Secventa alternativa:

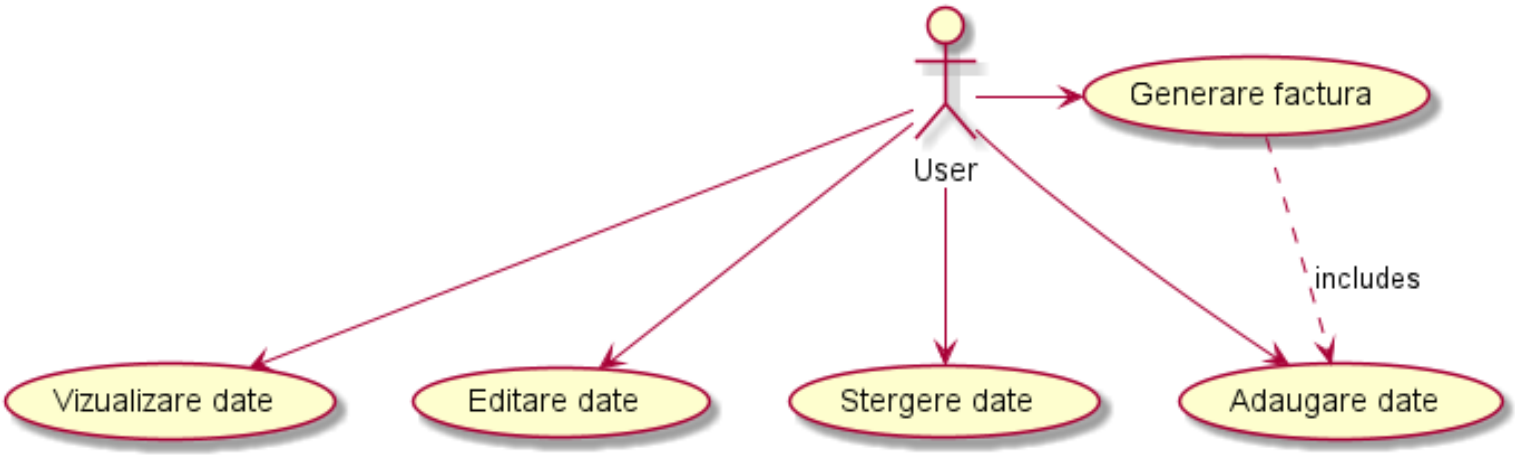
- In eventualitatea aparitiei unei erori, aceasta este afisata utilizatorului.
- Scenariul ajunge la pasul 1.

Caz de utilizare: stergerea datelor sau generarea facturii
Actorul principal: utilizatorul
Scenariul principal:

- Utilizatorul navigheaza la meniul dorit utilizand interfata grafica.
- Utilizatorul apasa butonul corespunzator operatiei care urmeaza sa fie efectuata.
- Un mesaj va fi afisat utilizatorului.
- Noile modificar i vor fi acum reflectate de interfata grafica.

Secventa alternativa:

- In eventualitatea aparitiei unei erori, aceasta este afisata utilizatorului.
- Scenariul ajunge la pasul 1.

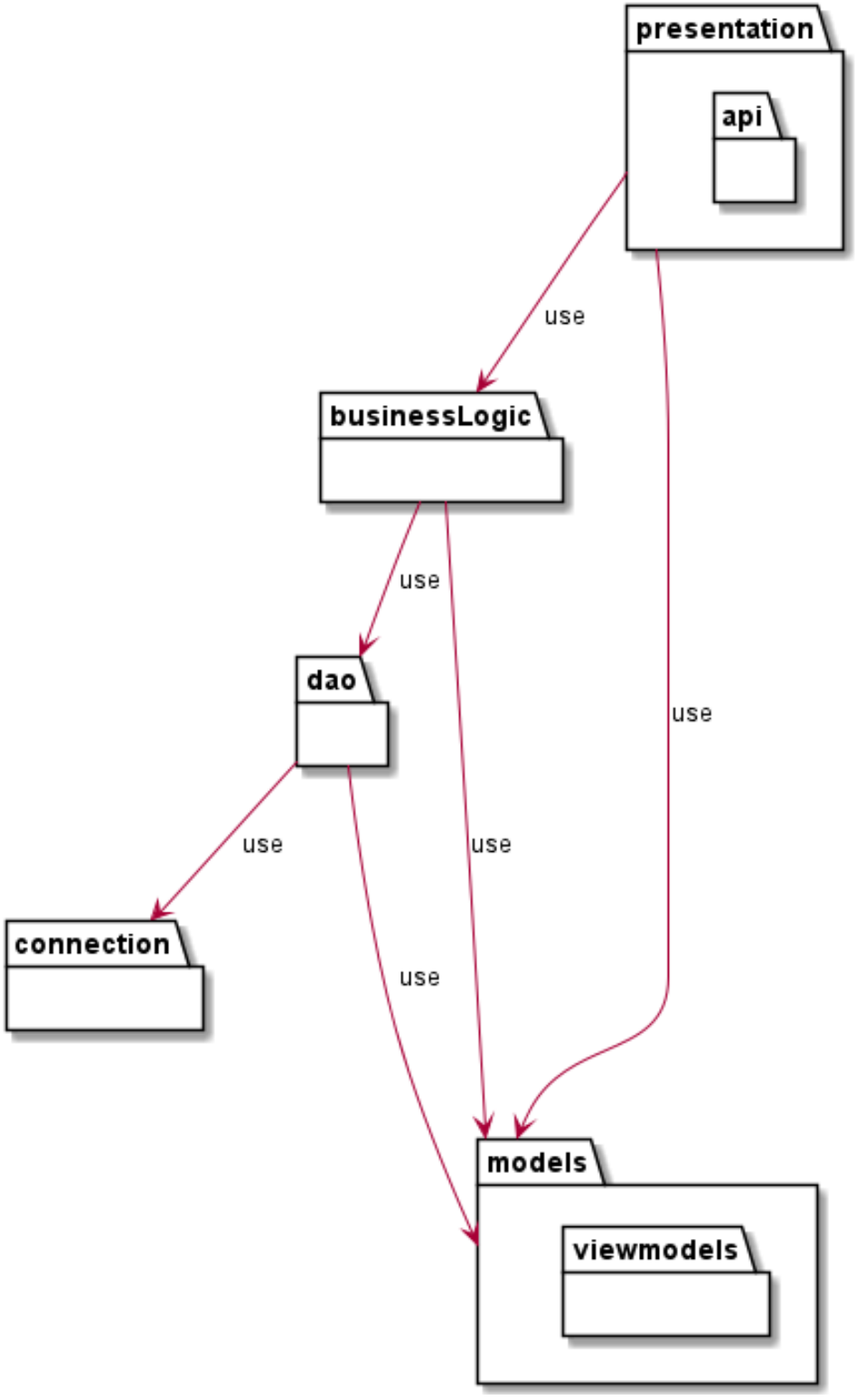


3. Proiectare

3.1. Diagrama de pachete

Am conceput acest proiect urmarind structura de pachete prezentata in figura de mai jos.

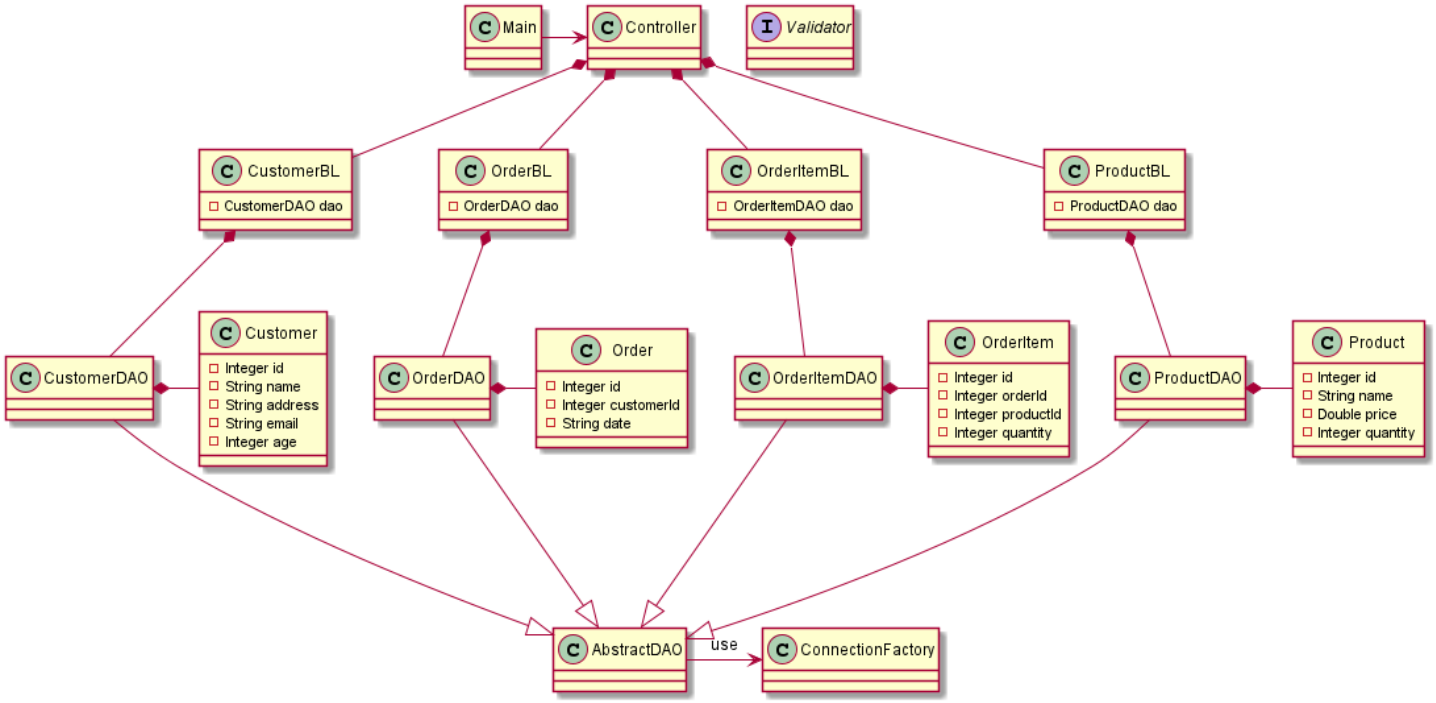
Se poate observa ca am utilizat un model arhitectural de tip stratificat (Layered Architecture) alaturi de un model de proiectare bazat pe strategia (Model View Controller).



Pachetele folosite sunt:

- Presentation
  - contine toate controllerele folosite in arhitectura MVC
- API
  - contine toate controllerele de tip REST
- Models
  - contine toate clasele care memoreaza date si / sau stari folosite in aplicatie
- ViewModels
  - contine toate clasele folosite in view-uri care retin date in vederea afisarii in interfata grafica
- BusinessLogic
  - contine toate clasele de baza de care aplicatia se foloseste pentru a lua decizii
- DAO
  - contine toate clasele folosite pentru a interactiona cu baza de date
- Connection
  - contine toate clasele cu functionalitate de sine statatoare utilizate in restul proiectului

3.2. Diagrama de clase

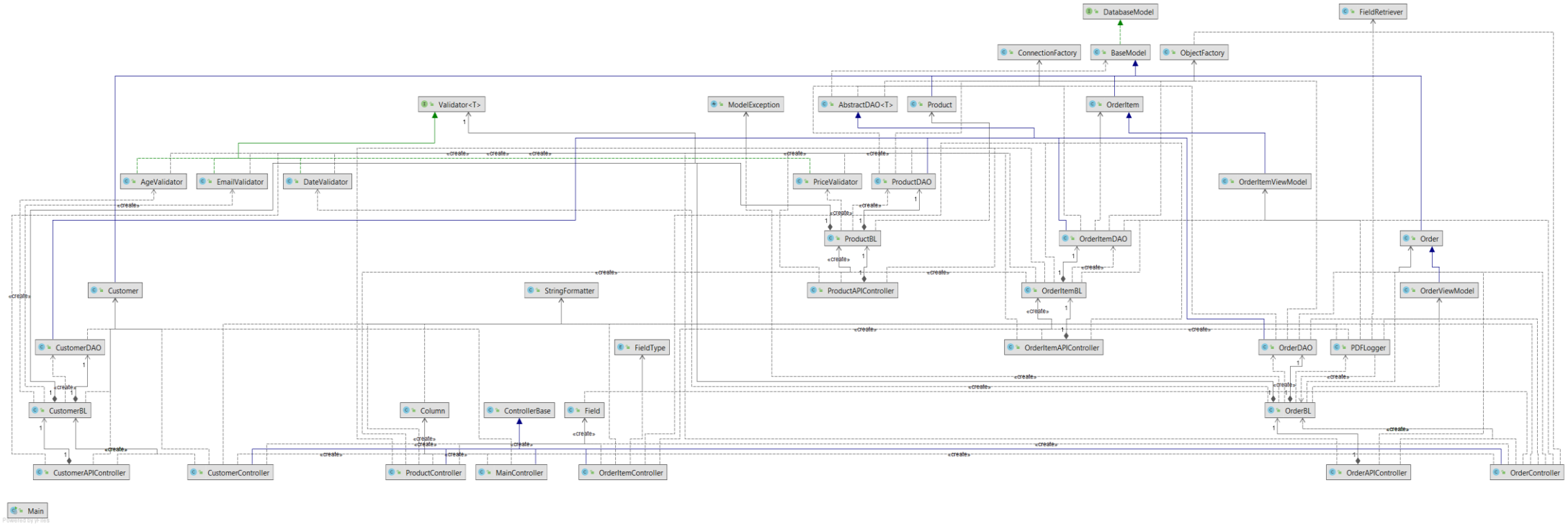


Clasele identificate in timpul procesului de proiectare a acestei teme de laborator sunt:

- Main
  - reprezinta punctul de start al aplicatiei
- Controller
  - este utilizata pentru a intercepta request-uri cauzate de catre utilizator si pentru a oferi un raspuns
- CustomerBL
  - contine logica necesara pentru a procesa datele despre un client
- OrderBL
  - contine logica necesara pentru a procesa datele despre o comanda
- OrderItemBL
  - contine logica necesara pentru a procesa datele despre un produs dintr-o comanda

- ProductBL
  - contine logica necesara pentru a procesa datele despre un produs
- CustomerDAO
  - se ocupa de operatii CRUD cu date despre clienti
- OrderDAO
  - se ocupa de operatii CRUD cu date despre comenzi
- OrderItemDAO
  - se ocupa de operatii CRUD cu date despre produse dintr-o comanda
- ProductDAO
  - se ocupa de operatii CRUD cu date despre produse
- AbstractDAO
  - se ocupa de operatii CRUD cu date din obiecte generice
- Customer
  - este utilizata pentru a retine date despre un client
- Order
  - este utilizata pentru a retine date despre o comanda
- OrderItem
  - este utilizata pentru a retine date despre un produs dintr-o comanda
- Product
  - este utilizata pentru a retine date despre un produs
- ConnectionFactory
  - este utilizata pentru initializa o conexiune cu baza de date

Aici avem diagrama de clase rezultata la finalul implementarii proiectului:



Interfetele definite sunt Validator si DatabaseModel. Aceste interfete au rolul atat de a desprinde functionalitatea de obiectul insusi, cat si pentru a asigura existenta atributelor dorite in interiorul modelelor folosite.

3.3. Structuri de date folosite

3.3.1. DriverManager

Structura de date DriverManager este o structura de date care ne ofera posibilitatea obtinerii unei conexiuni cu o baza de date.

Am folosit structura de date DriverManager pentru a genera conexiuni cu baza de date.

3.3.2. Connection

Structura de date Connection este o structura de date care reprezinta o sesiune cu baza de date.

Am folosit structura de date Connection pentru a executa diverse interogari pe baza de date.

3.3.3. PreparedStatement

Structura de date PreparedStatement este o structura de date care ne permite introducerea parametrilor in textul unei interogari intr-o maniera sigura pentru a evita diverse atacuri.

Am folosit structura de date PreparedStatement pentru a crea diverse interogari pentru baza de date.

3.3.4. ResultSet

Structura de date ResultSet este o structura de date care contine rezultatul executiei unei interogari in cadrul bazei de date.

Am folosit aceasta structura de date pentru a extrage rezultatele interogarilor executate pe baza de date.

3.4. Interfete definite

3.4.1. Validator

Interfata Validator are rol de a abstractiza logica de a valida un camp de implementarea acesteia.

3.4.2. DatabaseModel

Interfata DatabaseModel are scopul de a ne asigura ca modelele cu care lucrap pe baza de date au campul id.

4. Implementare

4.1. Descriere clase

Clasa MainMenuController

- Este componenta care gestioneaza interactiunea utilizatorului cu aplicatia si lucreaza cu modelul.
- Este C-ul din modelul arhitectural MVC.

private final MainMenuViewModel model;
- Obiect ce memoreaza starea curenta a aplicatiei si genereaza instante noi de simulare

private final MainMenuView view;
- Obiect ce contine interfata grafica (view-ul)

private void setUpUserEvents()
- Metoda ce atribuie componentelor din interfata grafica ascultatoare de evenimente (ActionListener)

Interfata Validator

- Este o interfata folosita pentru a separa logica de a valida un camp de implementarea acesteia.

Interfata DatabaseModel

- Este o interfata folosita pentru a ne asigura ca modelele folosite la lucrul cu baza de date au campul id.

Enumeratia FieldType

- Este o enumeratie care contine toate tipurile de camp care pot aparea intr-un formular in cadrul interfetei vizuale.

Clasa CustomerBL

- Clasa care se ocupa de logica necesara pentru procesarea datelor unui client.

```
private final CustomerDAO dao;
```
- Este un obiect ce se ocupa de operatii CRUD cu date despre clienti.

```
private final Validator ageValidator;
```
- Este un obiect ce se ocupa de validarea unei varste.

```
private final Validator emailValidator;
```
- Este un obiect ce se ocupa de validarea unui email.

Clasa OrderBL

- Clasa care se ocupa de logica necesara pentru procesarea datelor unei comenzi.

```
private final OrderDAO dao;
```
- Este un obiect ce se ocupa de operatii CRUD cu date despre comenzi.

```
private final Validator dateValidator;
```
- Este un obiect ce se ocupa de validarea unei date dupa formatul MM/dd/yyyy hh:mm a.

```
public void finalizeOrder(int id) throws SQLException, ModelException
```
- Metoda care finalizeaza comanda si genereaza un fisier PDF.

```
public List<OrderViewModel> findAllViewModels() throws SQLException,
ModelException
```
- Metoda care intoarce toate datele in vederea afisarii acestora in interfata grafica.

Clasa OrderItemBL

- Clasa care se ocupa de logica necesara pentru procesarea datelor unui produs dintr-o comanda.

```
private final OrderItemDAO dao;
```
- Este un obiect ce se ocupa de operatii CRUD cu date despre produse dintr-o comanda.

```
private void checkQuantity(OrderItem orderItem) throws SQLException,
ModelException
```
- Metoda care verifica cantitatea produsului dintr-o comanda sa fie pozitiva.

```
public List<OrderItemViewModel> findAllViewModelsByOrderId(int id) throws
SQLException, ModelException
```
- Metoda care intoarce toate datele dintr-o comanda in vederea afisarii acestora in interfata grafica.

Clasa ProductBL

- Clasa care se ocupa de logica necesara pentru procesarea datelor unui produs.

```
private final ProductDAO dao;
```
- Este un obiect ce se ocupa de operatii CRUD cu date despre produse.

```
private final Validator priceValidator;
```
- Este un obiect ce se ocupa de validarea unui pret.

```
public List<Product> findAllSelectableByOrderId(int id) throws SQLException,
ModelException
```
- Metoda care intoarce toate produsele ce pot fi introduse intr-o comanda in vederea afisarii acestora in interfata grafica.

Clasa CustomerDAO

- Clasa care se ocupa de operatii CRUD cu date despre clienti.

Clasa OrderDAO

- Clasa care se ocupa de operatii CRUD cu date despre comenzi.

```
public List<OrderViewModel> findAllViewModels() throws SQLException
```
- Metoda care intoarce toate datele in vederea afisarii acestora in interfata grafica.

Clasa OrderItemDAO

- Clasa care se ocupa de operatii CRUD cu date despre produse dintr-o comanda.

```
public List<OrderItemViewModel> findAllViewModelsByOrderId(int id) throws
SQLException
```
- Metoda care intoarce toate datele dintr-o comanda in vederea afisarii acestora in interfata grafica.

Clasa ProductDAO

- Clasa care se ocupa de operatii CRUD cu date despre produse.

```
public List<Product> findAllSelectableByOrderId(int id) throws SQLException
```

- Metoda care intoarce toate produsele ce pot fi introduse intr-o comanda in vederea afisarii acestora in interfata grafica.

Clasa AbstractDAO

- Clasa care se ocupa de operatii CRUD pe modele generice.

```
protected final Class<T> type;
```

- Obiect care indica clasa corespondenta tipului modelului folosit pentru efectuarea operatiilor CRUD.

```
public void insert(T obj) throws SQLException
```

- Metoda care insereaza datele despre modelul T in baza de date.

```
public void update(T obj) throws SQLException
```

- Metoda care actualizeaza datele despre modelul T in baza de date.

```
public void delete(T obj) throws SQLException
```

- Metoda care sterge datele despre modelul T in baza de date.

```
public void delete(int id) throws SQLException
```

- Metoda care insereaza datele despre modelul cu id-ul specificat in baza de date.

```
public T findById(int id) throws SQLException
```

- Metoda care extragele datele despre modelul cu id-ul specificat din baza de date.

```
public List<T> findAll() throws SQLException
```

- Metoda care extragele datele despre toate modelele din baza de date.

Clasa ConnectionFactory

- Clasa care se ocupa de crearea de conexiuni cu baza de date.

```
private static final Logger LOGGER;
```

- Obiect care ne afisarea unor informatii in consola sau intr-un fisier specificat.

```
private static final String DB_URL;
```

- String care indica adresa bazei de date.

```
private static final String USER;
```

- String care indica utilizatorul cu care se face logarea la baza de date.

```
private static final String PASS;
```

- String care indica parola cu care se face logarea la baza de date.

```
private static final ConnectionFactory instance;
```

- Obiect de tip singleton care creeaza conexiuni cu baza de date.

```
public static Connection getConnection() throws SQLException
```

- Metoda care creeaza o noua conexiune cu baza de date si o returneaza.

Exceptia ModelException

- Exceptie care indica faptul ca a aparut in timpul lucrului cu modele.

Clasa BaseModel

- Clasa de baza pentru un model care ne asigura existenta unui id.

Clasa Column

- Clasa care faciliteaza crearea unui header pentru tabelul din interfata vizuala.

Clasa Field

- Clasa care faciliteaza crearea unui camp de adaugat intr-un formular.

Clasa Customer

- Clasa care contine date despre un client.

```
private Integer id;
private String name;
private String address;
private String email;
private Integer age;
```

Clasa Order

- Clasa care contine date despre o comanda.

```
private Integer id;
private Integer customerId;
private String date;
```

Clasa OrderItem

- Clasa care contine date despre un produs dintr-o comanda.

```
private Integer id;
private Integer orderId;
private Integer productId;
private Integer quantity;
```

Clasa Product

- Clasa care contine date despre un produs.

```
private Integer id;
private String name;
private Double price;
private Integer quantity;
```

Clasa OrderItemViewModel & Clasa OrderViewModel

- Clasa care contine date suplimentare in vederea afisarii in interfata grafica.

Clasa AgeValidator & DateValidator & EmailValidator & PriceValidator

- Este o clasa ce implementeaza logica de validare a unui camp ce urmeaza sa fie introdus in baza de date.

```
@Override
boolean validate(T t) throws ModelException
```

- Metoda care adauga clientul intr-una din cozile specificate in functie de o regula predefinita.

Clasa FieldRetriever

- Clasa care faciliteaza obtinerea atributelor declarate.

```
public static List<Field> getFields(Class<?> clazz)
```

- Metoda care obtine si returneaza toate campurile declarate din ierarhia de mostenire a clasei specificate.

Clasa ObjectFactory

- Clasa care faciliteaza crearea obiectelor dintr-un rezultat provenit dintr-o interogare a bazei de date.

```
public static <T> List<T> createObjects(Class<T> clazz, ResultSet res) throws
Exception
```

- Metoda care creeaza si returneaza obiecte de tipul clasei specificate din rezultatul interogarii trimis ca parametru.

Clasa PDFLogger

- Clasa care faciliteaza crearea unui document PDF cu comenzile finalizate.

```
private final static String path = "./bills/";
```

- String ce reprezinta calea catre locul unde vor fi salvate documentele PDF.

```
public static void createBill(int id)
```

- Metoda care genereaza factura sub forma unui document PDF pentru comanda cu id-ul specificat.

Clasa StringFormatter

- Clasa care faciliteaza formatarea stringurilor.

```
public static String splitCamelCase(String s)
```

- Metoda care desparte cuvintele dintr-un string cu formatul camel case.

```
public static String capitalize(String s)
```

- Metoda care scrie cu majuscula prima litera a string-ului specificat.

Clasele

- **CustomerAPIController**
- **OrderAPIController**
- **OrderItemAPIController**
- **ProductAPIController**

- Clase care se ocupa de request-urile clientului pentru operatii de tip CRUD cu datele din baza de date.

Clasa ControllerBase

- Clasa care se ocupa de randarea unor view-uri partiale.

```
@Autowired
ViewResolver viewResolver;
```

- Obiect care faciliteaza randarea view-urilor.

```
protected String getView(String viewName, ModelMap modelMap)
```

- Metoda care returneaza view-ul specificat randat cu lista de parametri speificata.

Clasa MainController

- Clasa care se ocupa de servirea continutului principal al aplicatiei.

```
@GetMapping
public ModelAndView index(HttpSession session, Model model)
```

- Metoda care returneaza pagina principala a aplicatiei.

Clasele

- **CustomerController**
- **OrderController**
- **OrderItemController**
- **ProductController**

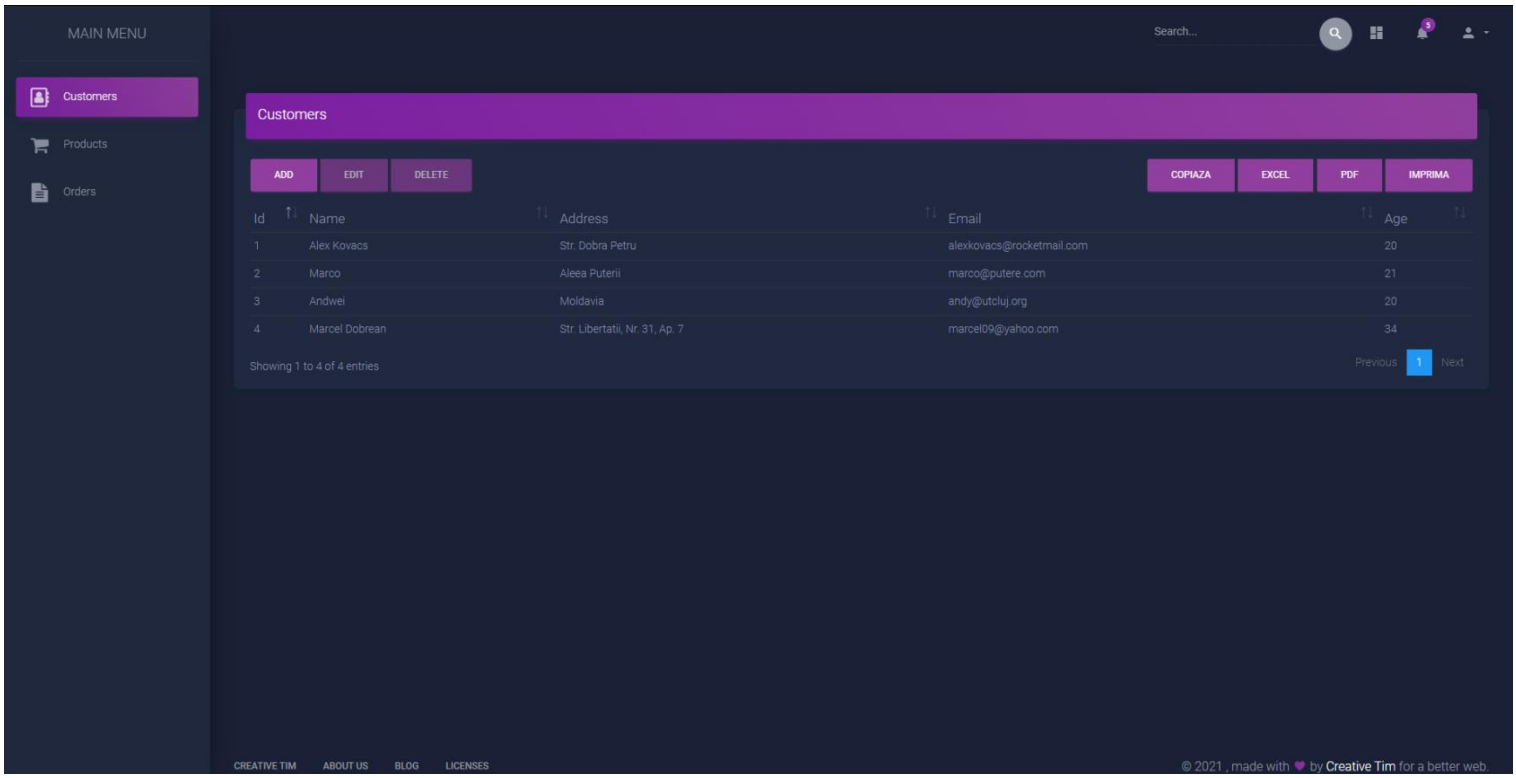
- Clase care se ocupa de request-urile clientului si returneaza noua pagina pe care interfata grafica urmeaza sa o prezinte.

Clasa Main

- Este o clasa statica in care se face initierea componentelor necesare pentru a completa modelul arhitectural MVC.
- Este punctul de start al aplicatiei.

4.2. Descriere implementare interfata utilizator

Interfata grafica este una user-friendly. Aceasta este usor de folosit, intuitiva si nu permite introducerea de date incorecte.



Am construit interfata grafica pornind de la un template deja existent. Pentru construirea interfetei grafice am folosit framework-ul Bootstrap 4. De asemenea, pentru tabele am folosit DataTables.

Interfata grafica a aplicatiei este alcatuita din trei pagini principale: una pentru clienti, una pentru produse si una pentru comenzi. Aceste 3 pagini au structura principala formata dintr-un tabel impreuna cu cateva butoane care ne ofera posibilitatea de a interactiona cu datele din tabel in diferite feluri. Ultima pagina difera de primele 2 prin faptul ca are in componenta sa un al doilea tabel care se afla intr-o relatie stransa cu primul.

4.2.1. Pagina pentru clienti

In cadrul paginii pentru clienti putem vedea datele despre clienti memorate in baza de date prin intermediul tabelului existent. Tot aici intalnim un numar de 3 butoane care ne permite sa interationam cu datele respective.

Cele 3 butoane prezente sunt:

- Butonul Add - ne permite adaugarea unui nou client in baza de date
- Butonul Edit - ne permite editarea datelor unui client din baza de date
- Butonul Delete - ne permite stergerea datelor despre un client din baza de date

Dupa apasarea primelor 2 butoane vom fi intampinati cu un formular unde trebuie sa introducem modificarile pe care dorim sa le incarcam in baza de date.

Butonul Delete va sterge la apasare linia selectata din tabel fara a mai cere aprobarea utilizatorului.

4.2.2. Pagina pentru produse

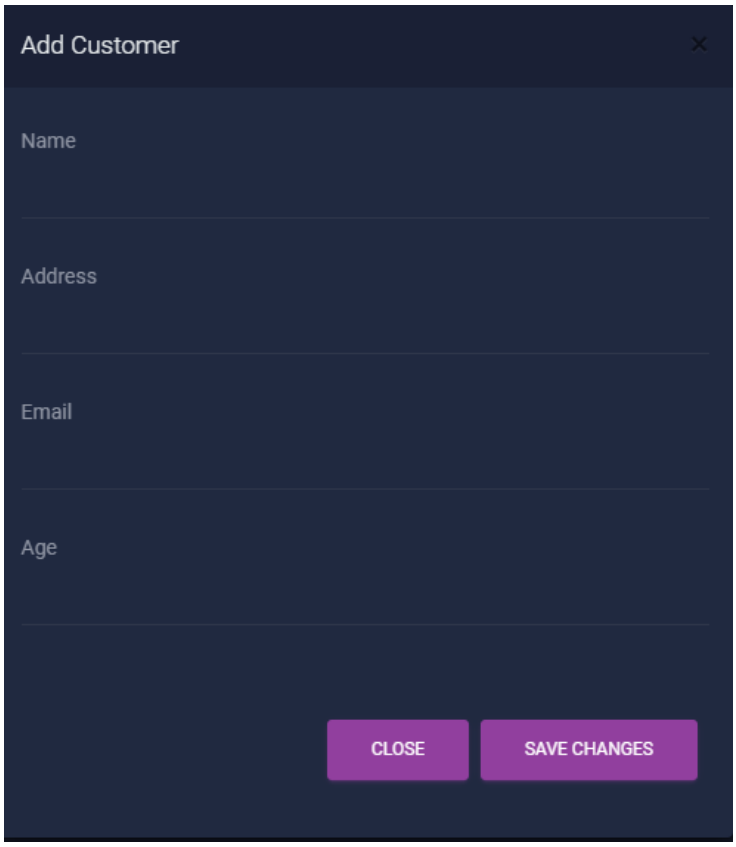
In cadrul paginii pentru produse putem vedea datele despre produse memorate in baza de date prin intermediul tabelului existent. Tot aici intalnim un numar de 3 butoane care ne permite sa interationam cu datele respective.

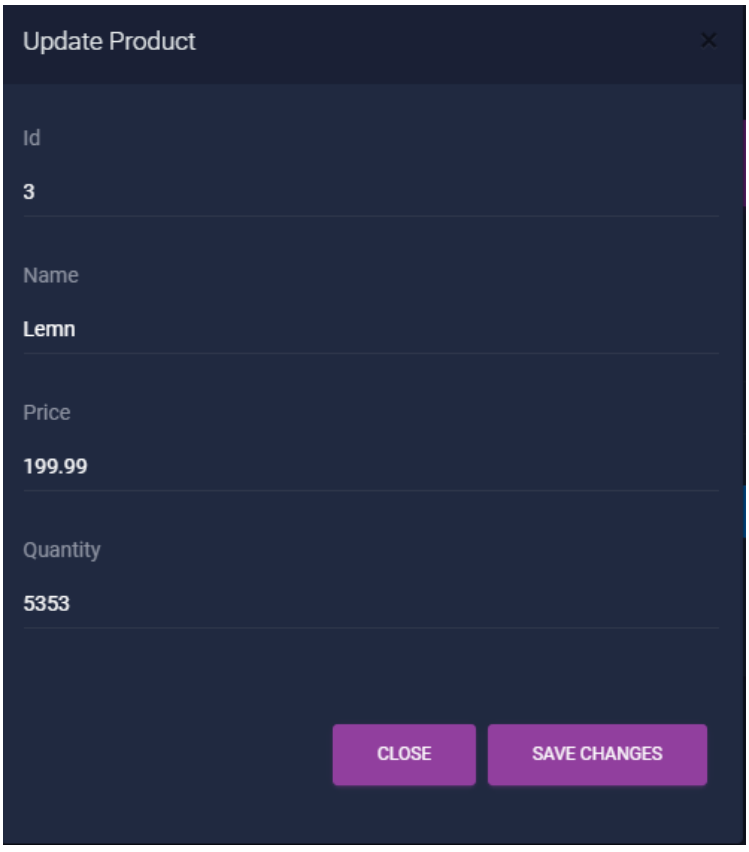
Cele 3 butoane prezente sunt:

- Butonul Add - ne permite adaugarea unui nou produs in baza de date
- Butonul Edit - ne permite editarea datelor unui produs din baza de date
- Butonul Delete - ne permite stergerea datelor despre un produs din baza de date

Dupa apasarea primelor 2 butoane vom fi intampinati cu un formular unde trebuie sa introducem modificarile pe care dorim sa le incarcam in baza de date.

Butonul Delete va sterge la apasare linia selectata din tabel fara a mai cere aprobarea utilizatorului.





4.2.3. Pagina pentru comenzi

In cadrul paginii pentru produse exista 2 tabele: unul pentru comenzi si unul pentru produsele din cadrul unei comenzi.

4.2.3.1. Tabelul pentru comenzi

In tabelul pentru comenzi putem vedea datele despre comenzile memorate in baza de date. Tot aici intalnim un numar de 4 butoane care ne permite sa interationam cu datele respective.

Cele 4 butoane prezente sunt:

- Butonul Add - ne permite adaugarea unei noi comenzi in baza de date
- Butonul Edit - ne permite editarea datelor unei comenzi din baza de date
- Butonul Delete - ne permite stergerea datelor despre o comanda din baza de date
- Butonul Finalize - ne permite finalizarea unei comenzi

Dupa apasarea primelor 2 butoane vom fi intampinati cu un formular unde trebuie sa introducem modificarile pe care dorim sa le incarcam in baza de date.

Butonul Delete va sterge la apasare linia selectata din tabel fara a mai cere aprobarea utilizatorului.

Butonul Finalize va finaliza comanda la apasare, daca este posibil, fara a mai cere aprobarea utilizatorului.

4.2.3.2. Tabelul pentru produse din cadrul comenzilor

In tabelul pentru comenzi putem vedea datele despre produsele din cadrul unor comenzi memorate in baza de date. Acest tabel devine vizibil doar cand este selectata o comanda din tabelul pentru comenzi. In tabel vor fi afisate mereu doar produsele din cadrul comenzii selectate.

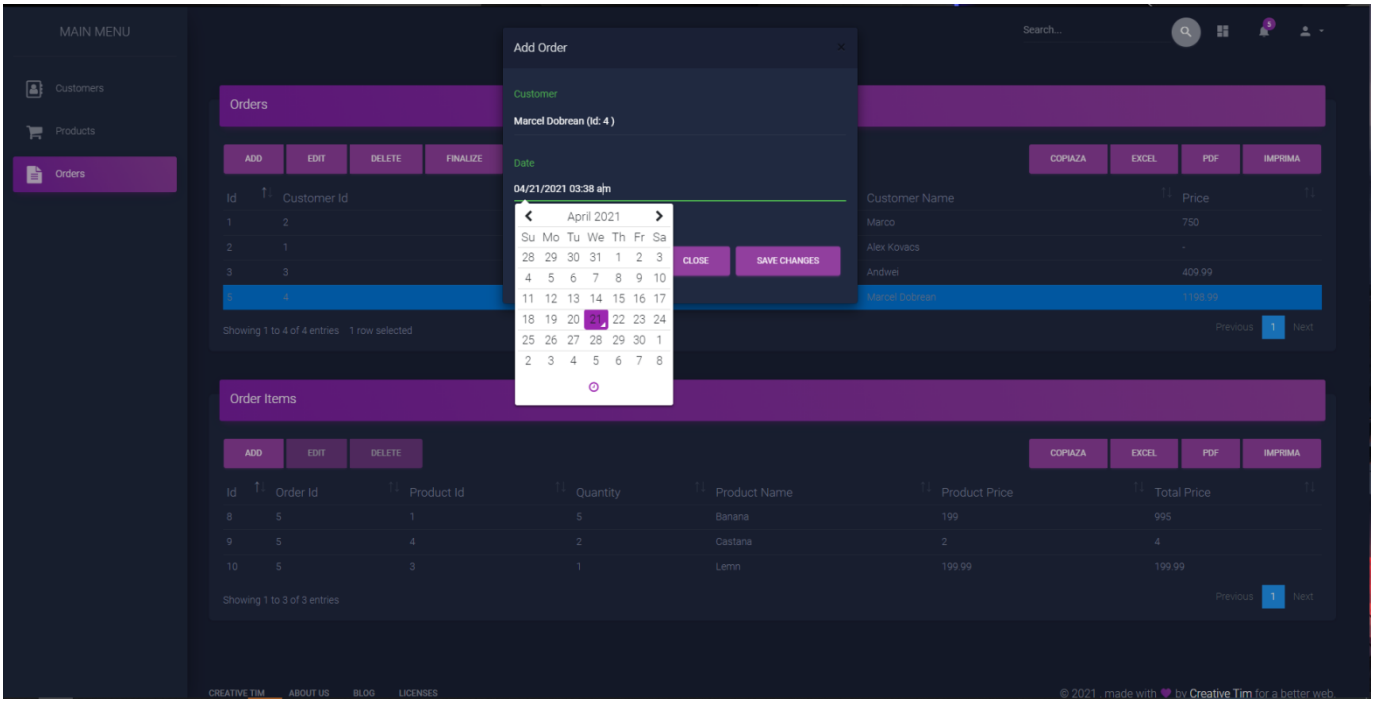
Tot aici intalnim un numar de 3 butoane care ne permite sa interationam cu datele respective.

Cele 3 butoane prezente sunt:

- Butonul Add - ne permite adaugarea unui nou produs in comanda in baza de date
- Butonul Edit - ne permite editarea datelor unui produs din comanda din baza de date
- Butonul Delete - permite stergerea datelor despre un produs din comanda din baza de date

Dupa apasarea primelor 2 butoane vom fi intampinati cu un formular unde trebuie sa introducem modificarile pe care dorim sa le incarcam in baza de date.

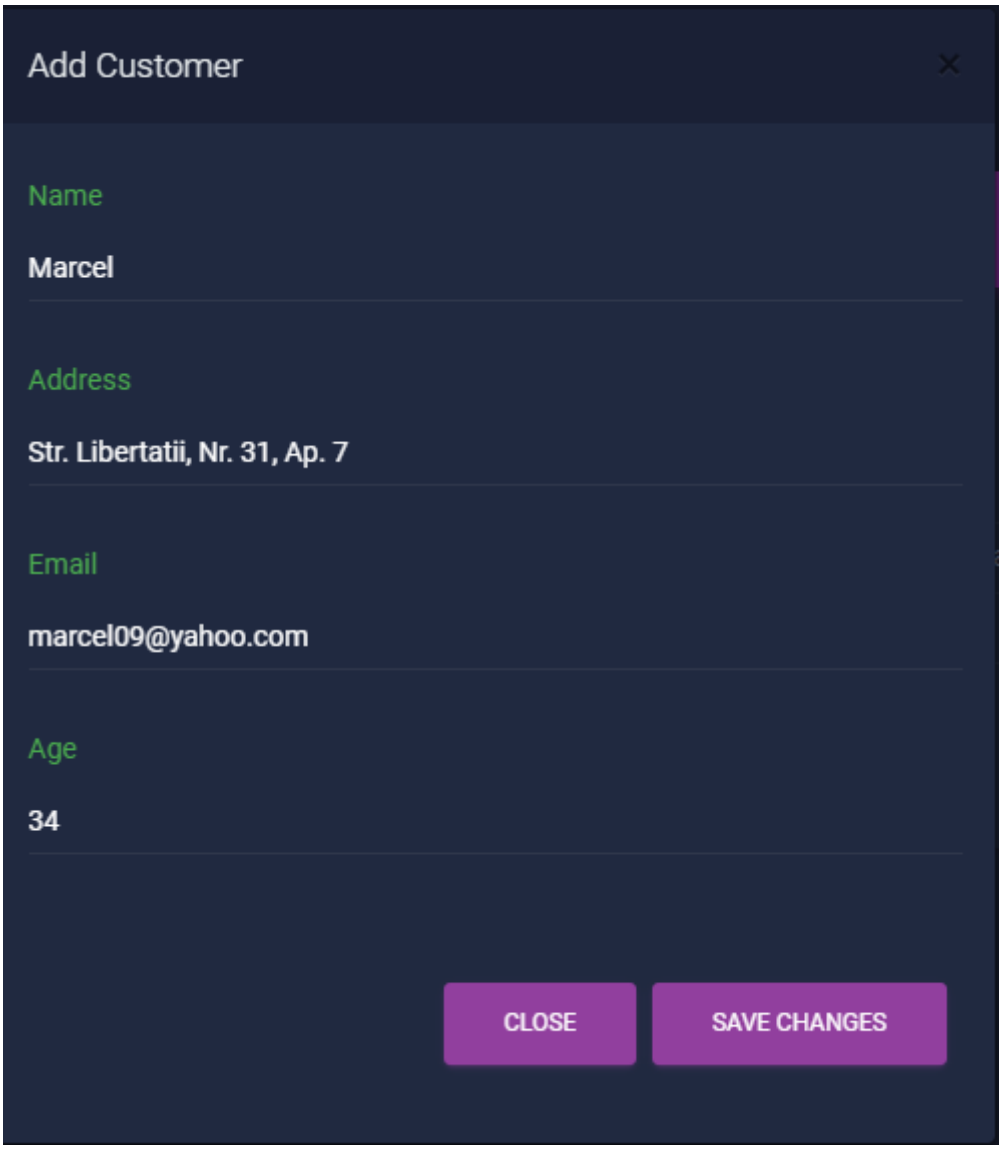
Butonul Delete va sterge la apasare linia selectata din tabel fara a mai cere aprobarea utilizatorului.



5. Rezultate

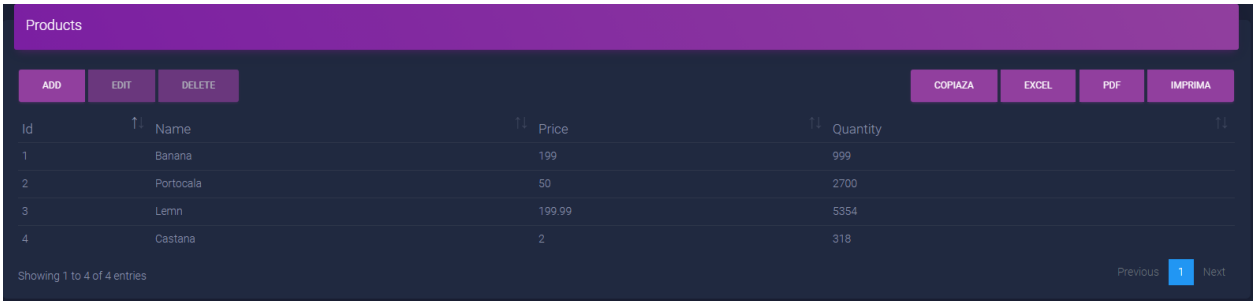
Pentru a verifica comportamentul aplicatiei de gestionare a datelor dintr-o baza de date am testat manual operatiile CRUD pentru fiecare model folosit in cadrul aplicatiei.

Scenariul 1 (Create)



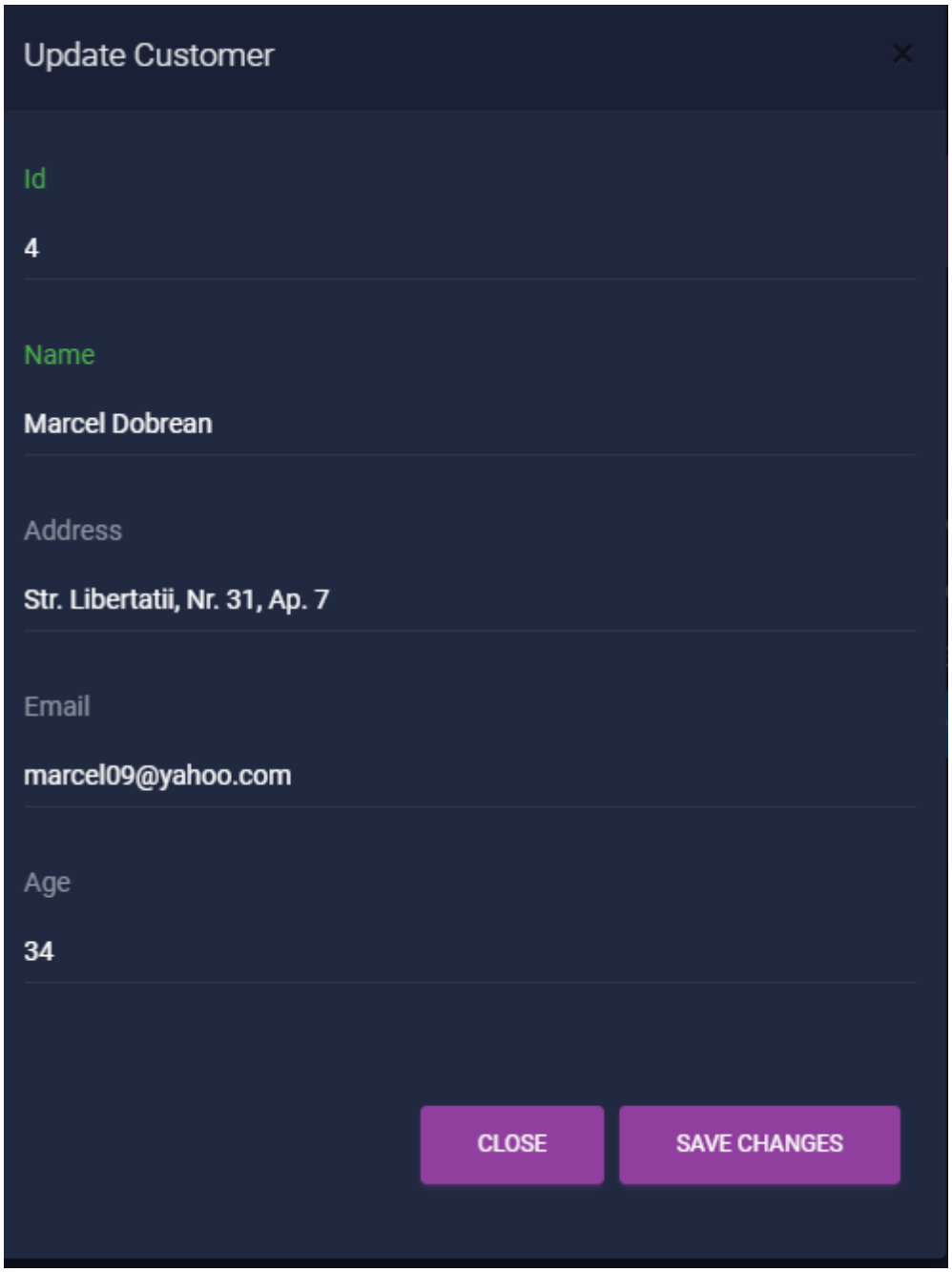
Actual result: Success
Expected result: Success
Trecut: Da

Scenariul 2 (Read)



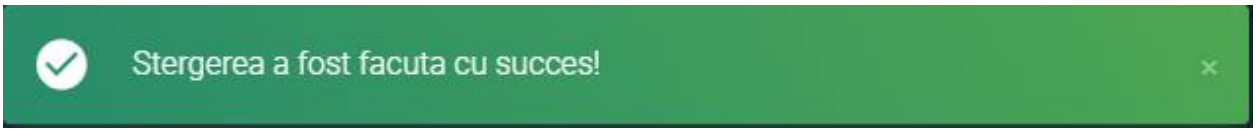
Actual result: Success
Expected result: Success
Trecut: Da

Scenariul 3 (Update)



Actual result: Success
Expected result: Success
Trecut: Da

Scenariul 4 (Delete)



Actual result: Success
Expected result: Success
Trecut: Da

6. Concluzii

Am realizat o aplicatie Java care faciliteaza administrarae unor date din cadrul unei baze de date prin intermerdiul unei interfete vizuale intuitive.

Acest proiect este unul pentru uz comun, se poate folosi oricand e nevoie de efectuarea unor operatii simple de tip CRUD asupra unei baze de date. Functioneaza fara probleme, au fost tratate diferite cazuri, astfel incat sa se obtina comportamentul dorit sau ca sa fie afisat un mesaj de eroare in cazul aparitiei unei erori.

In cadrul acestei teme am invatat sa folosesc Java Reflection, metode de a interactiona cu bazele de date, sa stapanesc mai bine folosirea modelului de proiectare Layered, sa lucrez cu design pattern-ul Singleton si Factory si sa acumulez mai multa experienta in utilizarea limbajului Java pentru a rezolva diverse probleme.

Ca dezvoltari ulterioare se pot aduce optimizari codului deja existent (modul de generare a formularelor), mai multe verificari asupra datelor care urmeaza sa fie introduse in baza de date, mai multe mesaje de eroare sau se pot implementa noi functionalitati precum posibilitatea de a modifica campurile retinute de fiecare model prin intermediul interfetei grafice sau chiar optiunea de a intoarce in timp datele din baza de date.

7. Bibliografie

- ✓ https://stackoverflow.com/
- ✓ JavaDoc
  - ➢ https://www.jetbrains.com/help/idea/working-with-code-documentation.html
  - ➢ https://www.baeldung.com/javadoc
- ✓ Java Reflection
  - ➢ ASSIGNMENT_3_SUPPORT_PRESENTATION
  - ➢ http://tutorials.jenkov.com/java-reflection/index.html
- ✓ LayeredArchitecture
  - ➢ https://dzone.com/articles/layers-standard-enterprise
- ✓ JDBC
  - ➢ https://www.javatpoint.com/example-to-connect-to-the-mysql-database
  - ➢ https://www.baeldung.com/java-jdbc
  - ➢ https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
- ✓ SQL dump file generation
  - ➢ https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html
- ✓ Diagrame
  - ➢ https://plantuml.com/sitemap-language-specification
- ✓ PDF
  - ➢ https://www.baeldung.com/java-pdf-creation
- ✓ Lombok
  - ➢ https://projectlombok.org/features/all
- ✓ Thymeleaf
  - ➢ https://stackoverflow.com/questions/25687816/setting-up-a-javascript-variable-from-spring-model-by-using-thymeleaf
- ✓ Template
  - ➢ https://demos.creative-tim.com/material-dashboard-dark/docs/2.0/getting-started/introduction.html
- ✓ Bootstrap
  - ➢ https://getbootstrap.com/docs/4.0/getting-started/introduction/
- ✓ DataTables
  - ➢ https://datatables.net/
- ✓ Java naming conventions
  - ➢ https://google.github.io/styleguide/javaguide.html