TEHNICI DE PROGRAMARE FUNDAMENTALE
ASSIGNMENT 3

FOOD DELIVERY MANAGEMENT SYSTEM

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 un sistem complex de gestiune a livrarii de alimente. Aceasta aplicatie admine 3 tipuri de utilizatori: administrator, angajat si client. Pentru fiecare din aceste tipuri permite efectuarea anumitor operatii. 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, se va genera o factura pentru fiecare comanda existenta care va fi salvata sub forma unui fisier TXT. Aplicatia se va folosi de o serializarea datelor local 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 sistemului de gestiune a livrarii de alimente – **Capitolul 3**
- Implementarea sistemului de gestiune a livrarii de alimente – **Capitolul 4**
- Testarea sistemului de gestiune a livrarii de alimente – **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:

- Sistemul proiectat ar trebui sa permita utilizatorilor de tip administrator efectuarea operatiilor:
  - Sa populeze initial meniul cu produsele dintr-un fisier
  - Sa efectueze operatii CRUD pe produse
  - Sa genereze rapoarte in functie de diferiti parametri
- Sistemul proiectat ar trebui sa permita utilizatorilor de tip angajat efectuarea operatiilor:
  - Sa fie anuntati cand se creeaza o noua comanda
  - Sa vizualizeze comenzile existente
- Sistemul proiectat ar trebui sa permita utilizatorilor de tip angajat efectuarea operatiilor:
  - Sa se inregistreze
  - Sa vada toate produsele existenta
  - Sa poata adauga produse in cos
  - Sa plaseze comanda
- Sistemul proiectat ar trebui sa genereze facturi pentru fiecare comanda efectuata
- Sistemul proiectat ar trebui sa notifice utilizator in cazul aparitiei unei erori

2.2. Cazuri de utilizare

Caz de utilizare: adaugarea si actualizarea produselor
Actorul principal: administratorul
Scenariul principal:

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

Secventa alternativa:

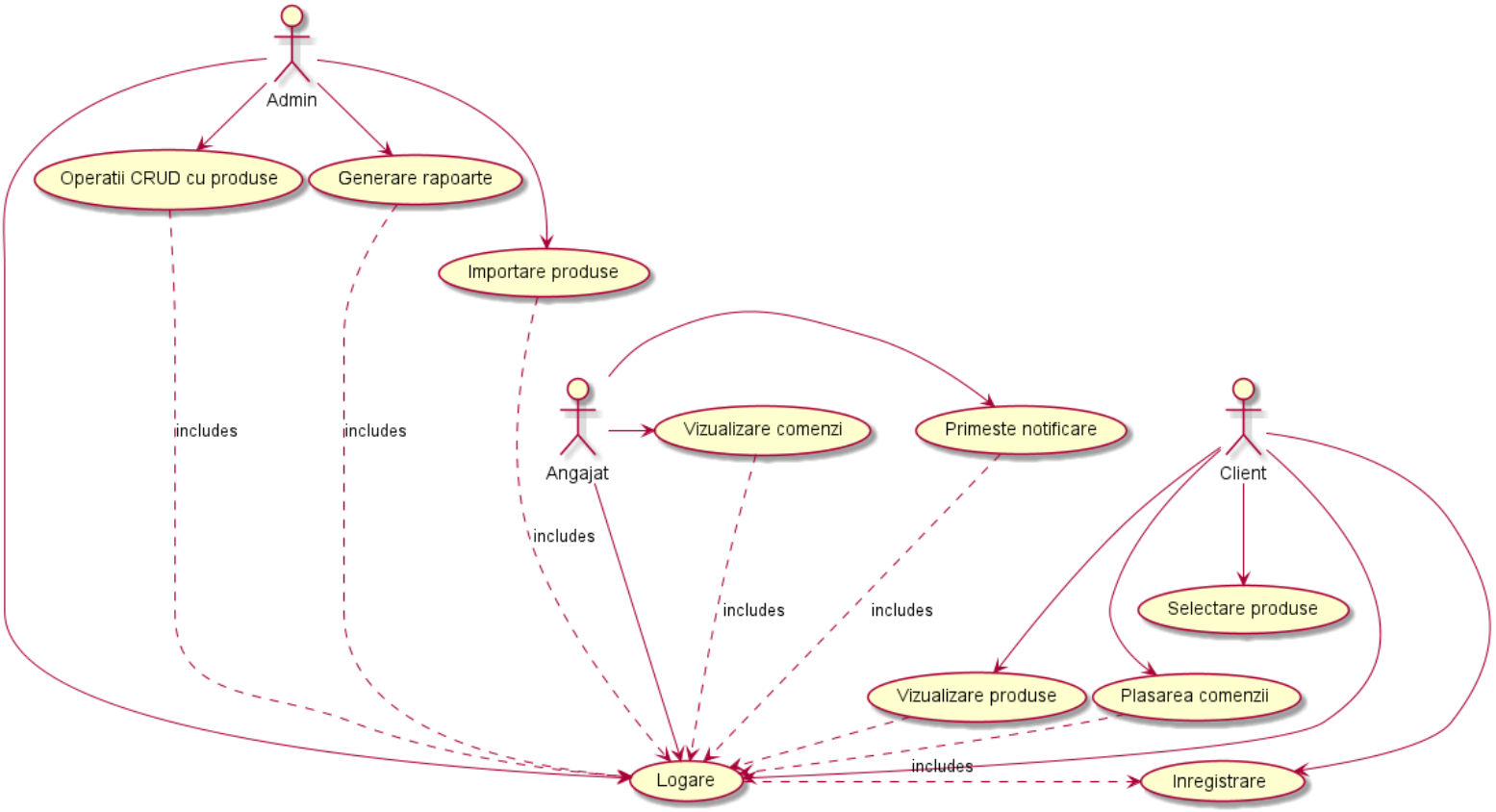
- In eventualitatea aparitiei unei erori, aceasta este afisata utilizatorului.
- Scenariul ajunge la pasul 1.

Caz de utilizare: plasarea comenzii
Actorul principal: clientul
Scenariul principal:

- Clientul navigheaza la meniul dorit utilizand interfata grafica.
- Clientul cauta si adauga in cos produsele dorite.
- Clientul navigheaza la meniul „Cart” utilizand interfata grafica.
- Clientul poate sterge produsele care nu mai sunt dorite cu butonul „Delete”
- Clientul apasa butonul „Order” pentru a plasa comanda.
- Un mesaj va fi afisat utilizatorului.
- Noile modificari vor fi acum reflectate de interfata grafica.

Secventa alternativa:

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

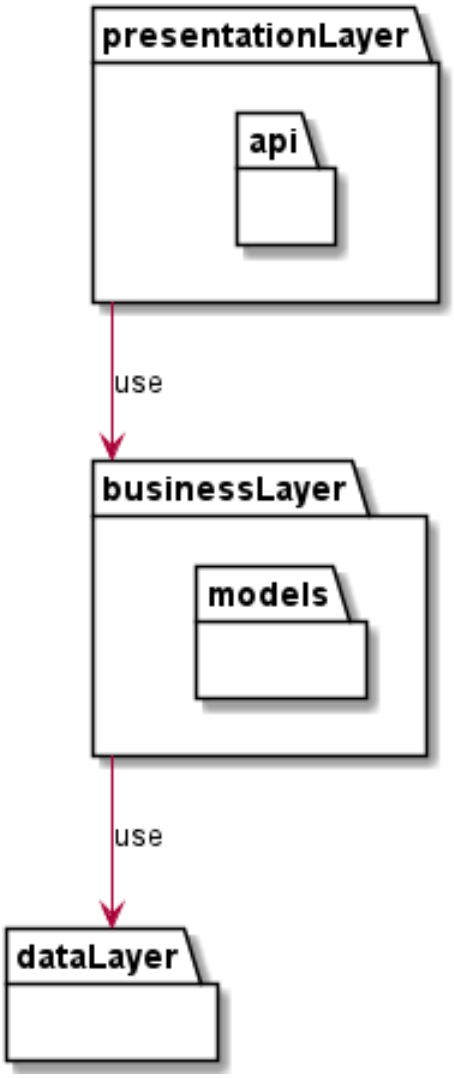


3. Proiectare

3.1. Diagrama de pachete

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

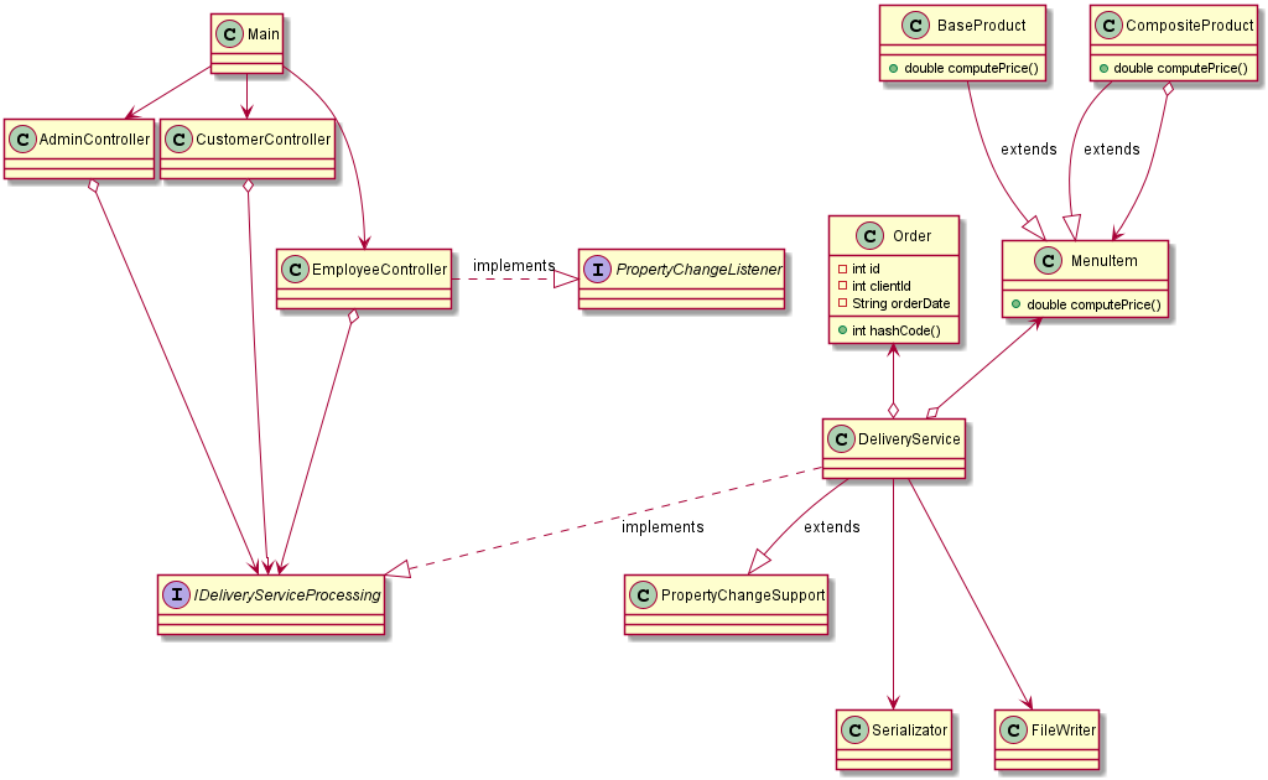
Se poate observa ca am utilizat un model arhitectural de tip stratificat (Layered Architecture) alaturi de un model de proiectare bazat pe strategia (Model View Controller).



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
  - contine toate clasele folosite in view-uri care retin date in vederea afisarii in interfata grafica
- Business
  - contine toate clasele de baza de care aplicatia se foloseste pentru a lua decizii
- Data
  - contine toate clasele folosite pentru a persista date

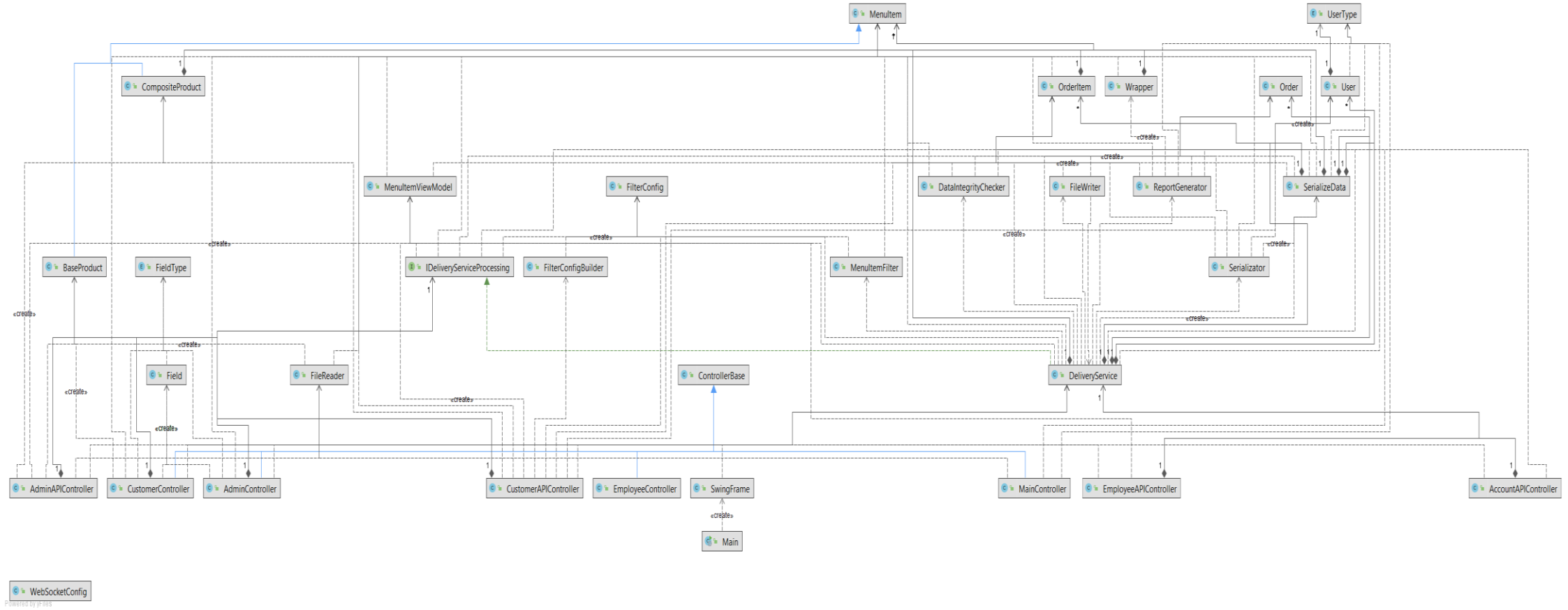
3.2. Diagrama de clase



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

- Main
  - reprezinta punctul de start al aplicatiei
- AdminController
  - este utilizata pentru a intercepta request-uri cauzate de catre administratori si pentru a oferi un raspuns
- CustomerController
  - este utilizata pentru a intercepta request-uri cauzate de catre clienti si pentru a oferi un raspuns
- EmployeeController
  - este utilizata pentru a intercepta request-uri cauzate de catre angajati si pentru a oferi un raspuns
- PropertyChangeListener
  - este utilizata pentru a implementa Observer Design Pattern (componenta Observer)
- PropertyChangeSupport
  - este utilizata pentru a implementa Observer Design Pattern (componenta Observable)
- IDeliveryServiceProcessing
  - contine logica necesara pentru a orchestra functionarea sistemului
- DeliveryService
  - contine implementarea logicii din IDeliveryServiceProcessing
- Order
  - memoreaza datele despre o comanda
- MenuItem
  - memoreaza datele de baza despre un produs din meniu
- BaseProduct
  - memoreaza datele despre un produs simplu din meniu
- CompositeProduct
  - memoreaza datele despre un produs compus din meniu
- Serializator
  - este utilizata pentru a serializa / deserializa date
- FileWriter
  - este utilizata pentru a scrie diverse date in fisiere

Aici avem diagrama de clase rezultata la finalul implementarii proiectului:



Singura interfetele definita este IDeliveryServiceProcessing. Aceasta interfeta are rolul de a abstractiza logica de a gestiona totalitatea sistemului de implementareaacesteia

3.3. Structuri de date folosite

3.3.1. HashMap

Structura de date HashMap este o structura de date care ne permite sa stocam si sa recuperam un obiect in timp constant O(1) cu conditia sa stim cheia corespunzatoare acestuia.

Am folosit structura de date HashMap pentru a stoca lista de produse dintr-o comanda in functie de comanda.

Comanda este cheia.

3.3.2. HashSet

Structura de date HashSet este o structura de date care memoreaza obiecte unice. Daca se incearca adaugarea unui duplicat, acesta va fi pur si simplu ignorat.

Am folosit structura de date HashSet pentru a obtine un set de produse unice in timpul importarii din fisier.

3.3.3. ArrayList

Structura de date HashSet este un sir de obiecte redimensionabil, care se comporta ca si o lista.

Am folosit structura de date ArrayList pentru a memora diverse date cu caracter enumerativ si/sau iterabil.

3.3.4. Stream

Structura de date Stream este, de fapt, o interfata. Aceasta interfata se refera la un set de obiecte a caror procesare se poate face atat secvential cat si in mod paralel.

Am folosit structura de date Stream pentru a procesa datele mai eficient.

3.4. Interfete definite

3.4.1. IDeliveryServiceProcessing

InterfetaIDeliveryServiceProcessing are rolul de a abstractiza logica de a gestiona totalitatea sistemului de implementareaacesteia.

4. Implementare

4.1. Descriere clase

Clasa SwingFrame

- Este componenta care permite inchiderea aplicatiei si persistarea datelor.

Interfata IDeliveryServiceProcessing

- Este o interfata care abstractizeaza logica sistemului pentru serviciul de livrare de implementarea acestuia.

```
void importItems(List<MenuItem> menuItems);
List<MenuItem> getMenuItems();
MenuItem getMenuItem(int id);
void addMenuItem(MenuItem item);
void updateMenuItem(int id, MenuItem item);
void removeMenuItem(int id);
boolean containsProduct(String title);
void generateReportBetweenHours(LocalTime startHour, LocalTime endHour);
void generateReportProducts(int times);
void generateReportClientSum(int minNoOrders, double minSum);
void generateReportProductsInDay(LocalDate date);
List<MenuItem> filterMenuItems(FilterConfig config);
void createOrder(User client, List<MenuItemViewModel> items);
List<Order> getOrders();
User checkAccount(User user);
boolean isWellFormed();
```

Clasa DeliveryService

- Este o clasa care se ocupa de intretinerea sistemului pentru serviciul de livrare.

```
private PropertyChangeSupport pcs;
   - Obiectul observable.
private List<MenuItem> menuItems;
   - Lista de produse.
private Map<Order, List<OrderItem>> orders;
   - Comenzile cu listele de produse comandate.
private List<User> accounts;
   - Lista de utilizatori.
public static final DeliveryService instance = new DeliveryService();
   - Obiectul de tip singleton.
private void init()
   - Metoda care initializeaza obiectul de tip singleton.
```

Enumeratia FieldType

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

Enumeratia UserType

- Este o enumeratie care contine toate tipurile de utilizator prezente in aplicatie.

Clasa Field

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

Clasa FilterConfig

- Clasa care faciliteaza filtrarea produselor.

Clasa SerializeData

- Clasa care faciliteaza transmiterea datelor serializate / de serializat.

Clasa Wrapper

- Clasa care faciliteaza crearea reporturilor.

Clasa MenuItem

- Clasa care contine date de baza despre un produs.

```
private int id;
private String title;
private double rating;
```

Clasa BaseProduct

- Clasa care contine date despre un produs de baza.

```
private double calories;
private double protein;
private double fat;
private double sodium;
private double price;
```

Clasa CompositeProduct

- Clasa care contine date despre un produs compus.

```
private List<MenuItem> items;
```

Clasa MenuItemViewModel

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

Clasa Order

- Clasa care contine date despre o comanda.

```
private Integer id;
private Integer clientId;
private String date;
```

Clasa OrderItem

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

```
private final MenuItem item;
private final int quantity;
```

Clasa User

- Clasa care contine date despre un utilizator.

```
private String username;
private String password;
private UserType type;
```

Clasa FileReader

- Clasa care faciliteaza filtrarea produselor.

```
public static List<MenuItem> getProducts(InputStream inputStream) throws
Exception
```

- Metoda care genereaza importa lista de produse.

```
private static BaseProduct mapToProduct(String[] arr)
```

- Metoda care transforma o lista de proprietati intr-un BaseProduct.

Clasa FileWriter

- Clasa care faciliteaza scrierea in fisiere.

```
public static void generateBill(Order order)
```

- Metoda care genereaza factura sub forma unui document TXT pentru comanda specificata.

```
public static void saveReport(String path, String content)
```

- Metoda care salveaza reportul la calea specificata.

Clasa Serializator

- Clasa care serializarea si deserializarea datelor in vederea persistarii acestora.

```
private static String path = "./data/serialized.data";
```

- String ce reprezinta calea catre locul unde vor fi serializate datele.

```
public static SerializeData deserialize(String path) throws Exception
```

- Metoda care deserializeaza datele de la calea specificata.

```
public static void serialize(String path, SerializeData data) throws Exception
```

- Metoda care serializeaza datele transmise la calea specificata.

Clasa DataIntegrityChecker

- Clasa care faciliteaza verificarii invariantului.

```
public static boolean isWellFormed(List<MenuItem> menuItems, Map<Order,
List<OrderItem>> orders, List<User> accounts)
```

- Metoda care verifica daca datele specificate respecta conditia invariantului.

Clasa FilterConfigBuilder

- Clasa care faciliteaza obtinerea unui filtru.

```
public FilterConfig getConfig()
```

- Metoda care genereaza un nou filtru.

Clasa MenuItemFilter

- Clasa care faciliteaza filtrarea produselor.

```
public static List<MenuItem> filterMenuItems(List<MenuItem> menuItems,
FilterConfig config)
```

- Metoda care filtreaza produsele conform conditiilor specificate in configuratia filtrului.

Clasa ReportGenerator

- Clasa care faciliteaza generarea reporturilor.

```
public static String generateReportBetweenHours(List<Order> orders, LocalTime
startHour, LocalTime endHour)
```

- Metoda care genereaza primul report din cerinta.

```
public static String generateReportProducts(List<MenuItem> menuItems, Map<Order,
List<OrderItem>> orders, int times)
```

- Metoda care genereaza al doilea report din cerinta.

```
public static String  generateReportClientSum(List<User> accounts, Map<Order,
List<OrderItem>> ordersMap, int minNoOrders, double minSum)
```

- Metoda care genereaza al treilea report din cerinta.

```
public static String generateReportProductsInDay(List<MenuItem> menuItems,
Map<Order, List<OrderItem>> ordersMap, LocalDate date)
```

- Metoda care genereaza al patrulea report din cerinta.

Clasele

- **AccountAPIController**
- **AdminAPIController**
- **CustomerAPIController**
- **EmployeeAPIController**

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

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

- **AdminController**
- **OrderController**
- **CustomerController**
- **EmployeeController**

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

Clasa WebSocketConfig

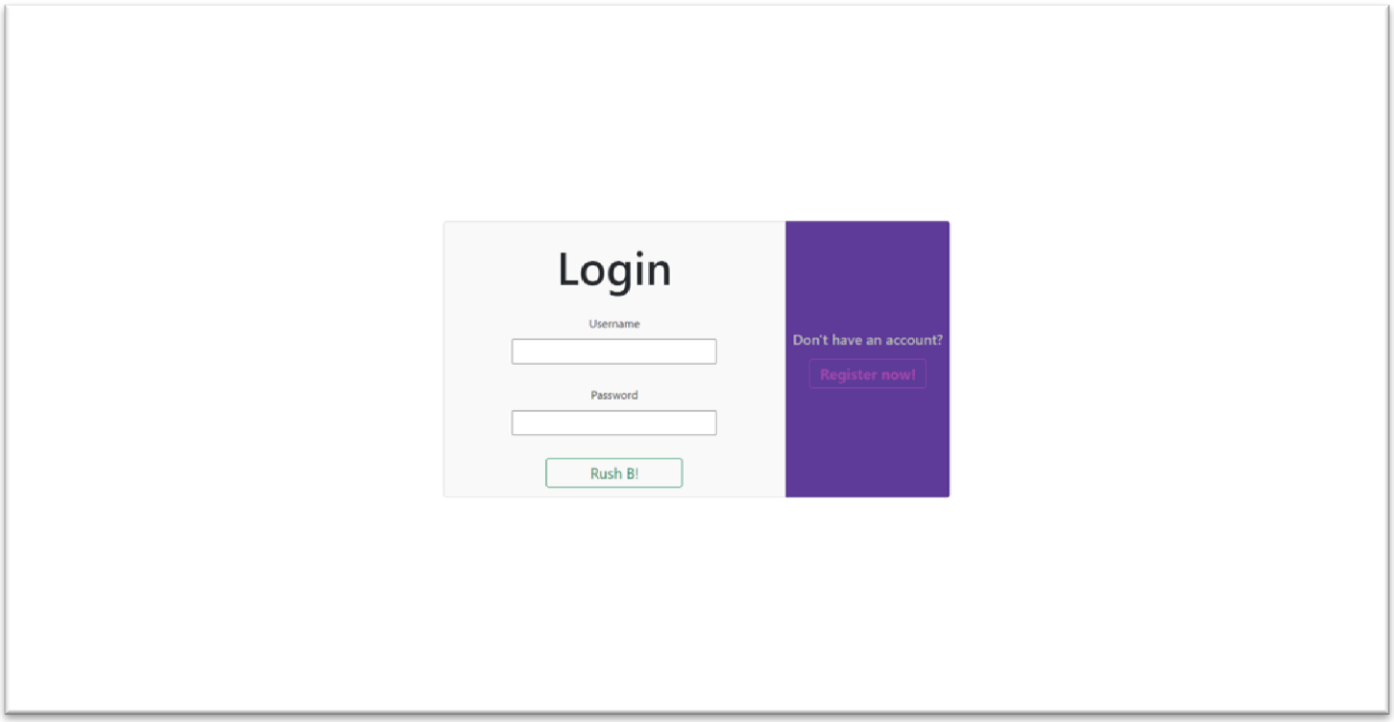
- Este o clasa folosita de Spring pentru configurarea serviciului de websockets folosit pentru notificarea angajatilor.

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 o interfata grafica noua. 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 administratori, una pentru angajati si una pentru clienti. 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 moduri diferite.

4.2.1. Pagina pentru administratori

In cadrul paginii pentru administratori putem vedea datele despre produsele memorate in cadrul aplicatiei prin intermediul unui tabel, putem importa setul initial de produse si putem genera diferite rapoarte in functie de parametri specificati. Aceste functionalitati sunt puse la dispozitie utilizatorului prin tab-urile din meniul de navigare din partea stanga.

Cele 3 tab-uri prezente sunt:

- Import - ne permite importarea setului principal de produse prin selectarea unui fisier csv
- Manage - ne permite efectuarea de operatii CRUD asupra produselor
- Reports - ne permite generarea rapoartelor

In cadrul tab-ului de „Import” putem selecta fisierul cu produse folosind butonul „Choose File”. Pentru a importa in sistem se foloseste butonul „Import!”.

In cadrul tab-ului „Manage” putem efectua operatii CRUD asupra produselor folosind butoanele corespunzatoare puse la dispozitie.

In cadrul tab-ului „Reports” putem selecta tipul raportului pe care vrem sa il generam folosind cele 4 butoane puse la dispozitie. Dupa apasarea unui buton vom fi intampinati cu un formular unde trebuie sa introducem parametri in functie de care vrem sa generam raportul.

4.2.2. Pagina pentru angajati

In cadrul paginii pentru angajati putem vedea datele despre comenzile efectuate pana in momentul curent. In plus, daca se plaseaza o comanda in timp ce suntem in fereastra de angajati vom fi anuntati cu o mica notificare si se va actualiza lista de comenzi. Aici intalnim doar un tab.

Tab-ul prezent este:

- Orders - ne permite vizualizarea comenzilor plasate

In cadrul tab-ului precizat putem afla mai multe informatii despre comanda si continutul acesteia cu ajutorul butonului „Detalii”.

4.2.3. Pagina pentru clienti

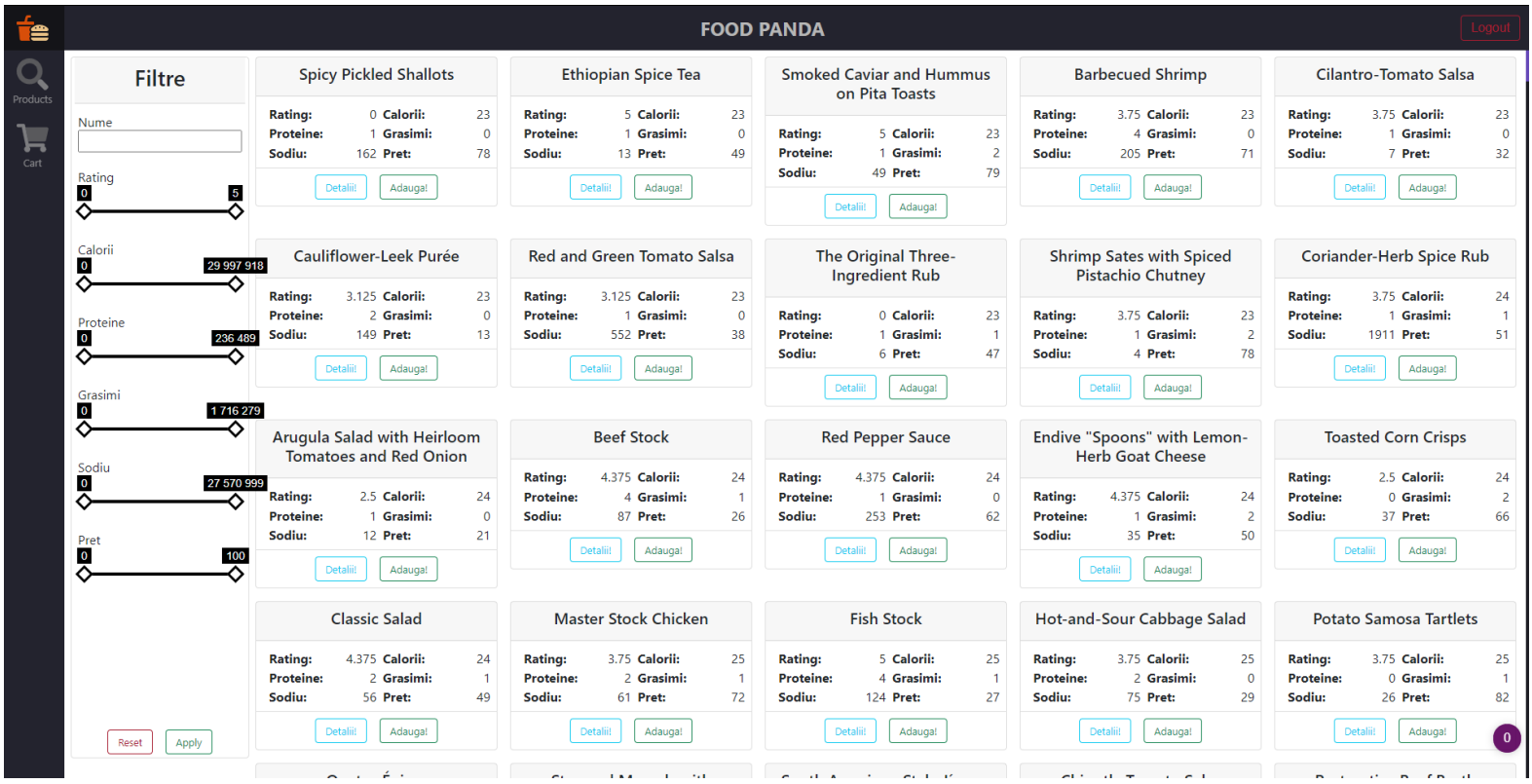
In cadrul paginii pentru clienti putem vedea totalitatea produselor puse la dispozitie de aplicatia noastra si putem plasa o comanda cu diverse produse. Aceste functionalitati sunt puse la dispozitie utilizatorului prin tab-urile din meniul de navigare din partea stanga.

Cele 2 tab-uri prezente sunt:

- Products - ne permite vizualizarea, filtrarea si selectarea produselor
- Cart - ne permite scoaterea produselor din cos si plasarea comenzii

In cadrul tab-ului „Products” putem vizualiza totalitatea produselor din meniu. Aici putem afla mai multe detalii despre un produs folosind butonul „Detalii!”. Prin intermediul butonului „Adauga!” putem adauga in cos produsul corespunzator. De asemenea, putem filtra produsele folosind meniul de filtrare. In acest meniu ne sunt puse la dispozitie mai multe filtre care pot fi aplicate pe selectia produselor. Dupa selectarea filtrelor dorite se apasa pe butonul „Apply!” si produsele vor fi filtrate. Stergerea filtrelor se poate face cu ajutorul butonului „Reset”.

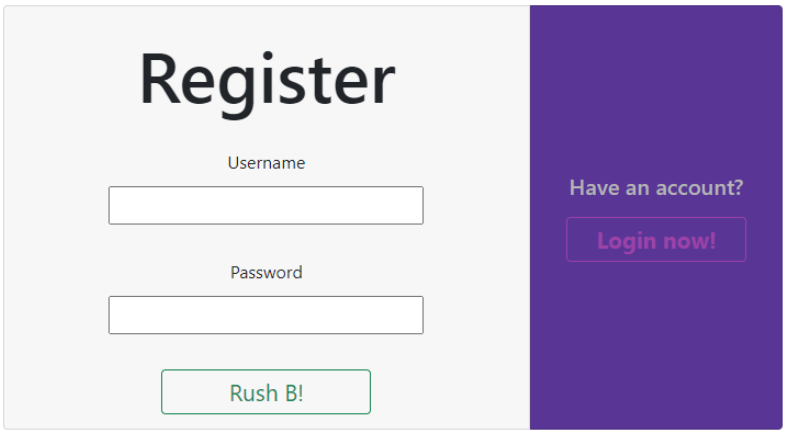
In cadrul tab-ului de „Cart” putem vizualiza produsele adaugate in cosul de cumparatori, putem sterge produse din cos si putem plasa comanda prin intermediul butoanelor corespunzatoare existente pe pagina.

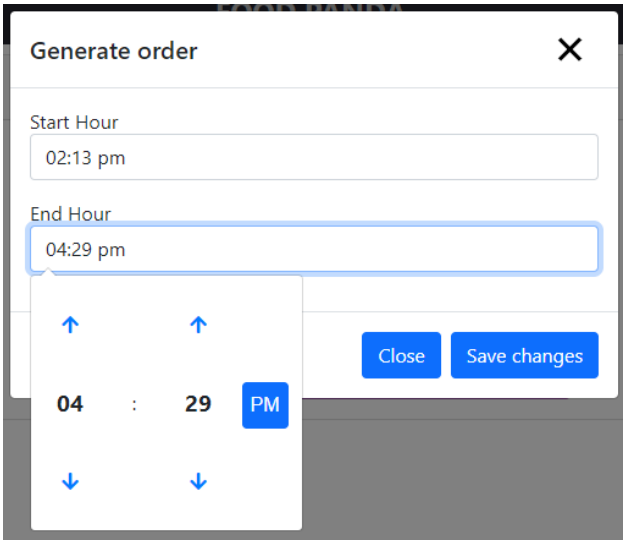


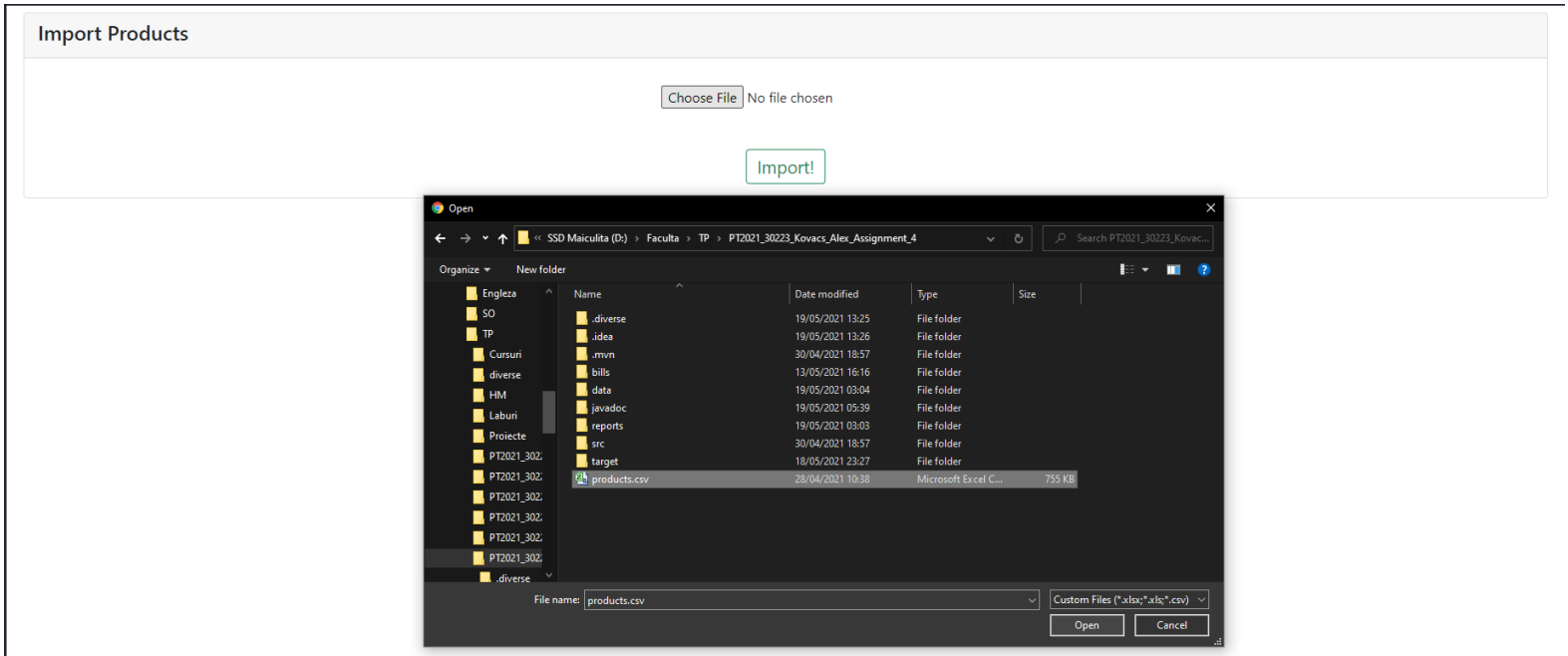
4.2.4. Pagina pentru inregistrare

In cadrul paginii pentru inregistrare putem crea noi conturi de tip client pentru a utiliza aplicatia. Daca datele introduse sunt valide autentificarea va fi facuta automat.

Daca dorim sa schimbam contul cu care suntem logati putem folosi butonul „Logout” pentru a ne deconecta.



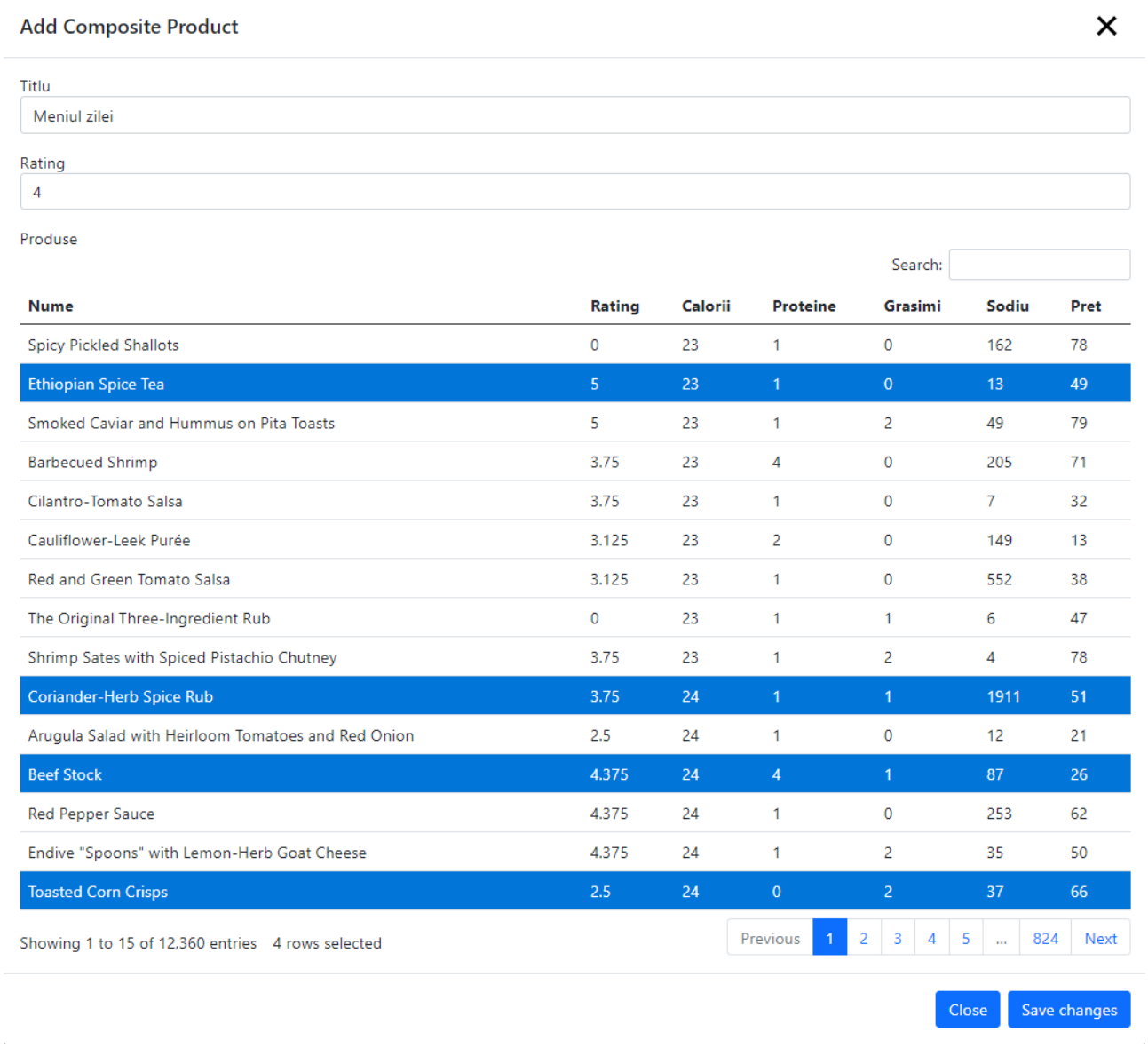




5. Rezultate

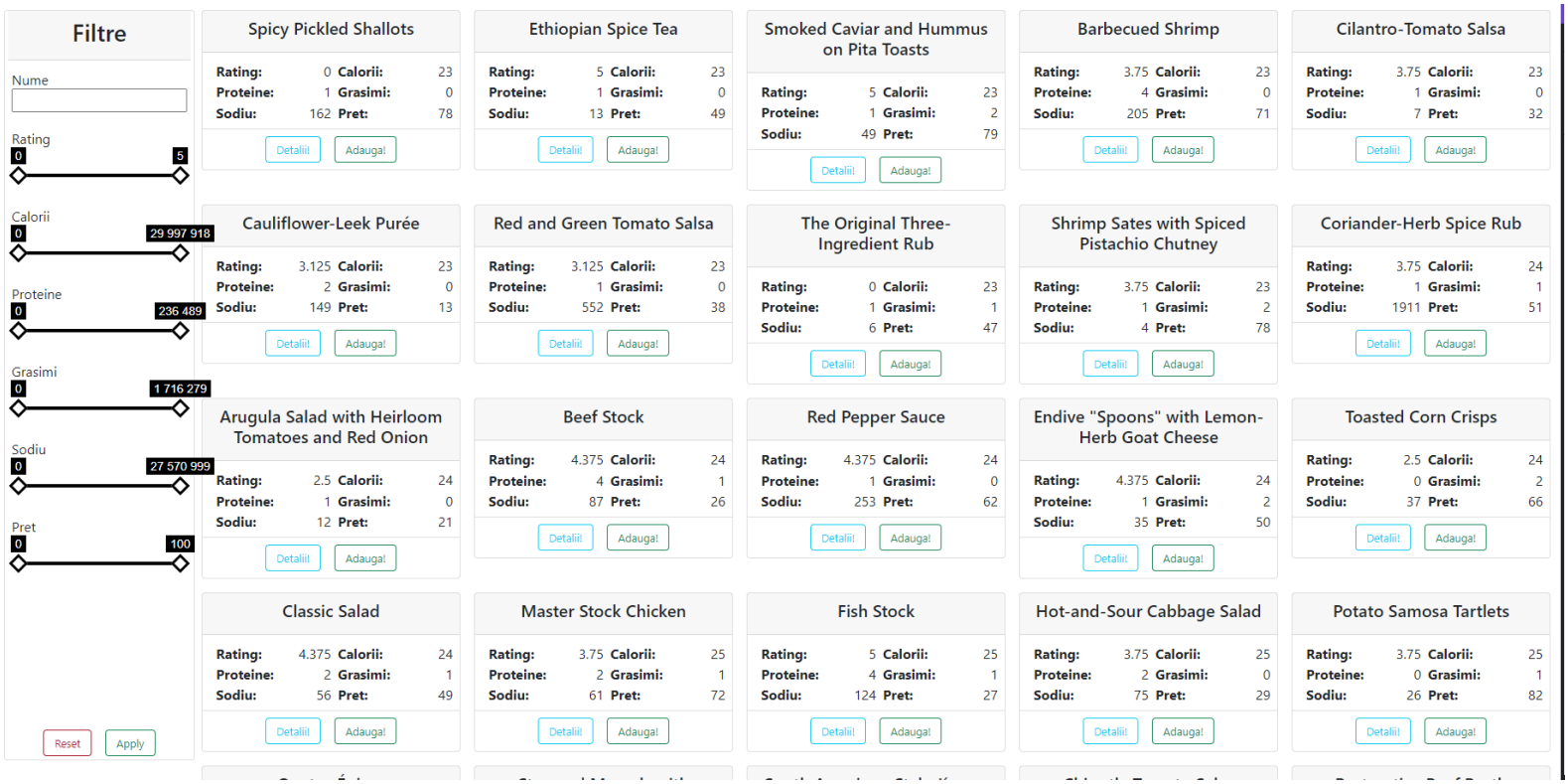
Pentru a verifica comportamentul aplicatiei de gestionare a datelor dintr-o baza de date am testat manual operatiile CRUD pentru fiecare model folosit in cadrul aplicatiei.

Scenariul 1 (Create)



Actual result: Success
Expected result: Success
Trecut: Da

Scenariul 2 (Read)

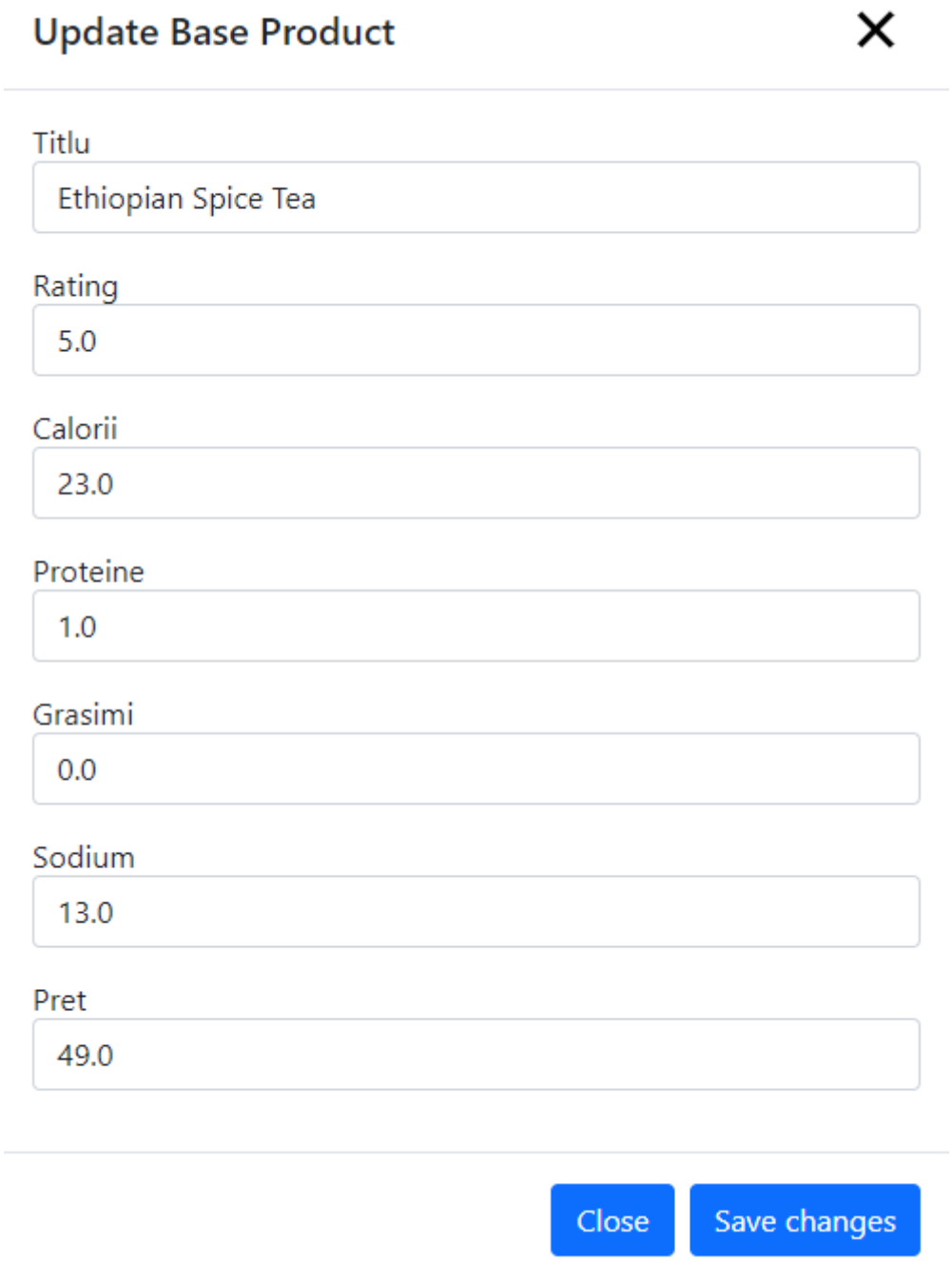


Actual result: Success
Expected result: Success
Trecut: Da

Scenariul 3 (Delete)

Actual result: Success
Expected result: Success
Trecut: Da

Scenariul 4 (Update)



Actual result: Success
Expected result: Success
Trecut: Da

6. Concluzii

Am realizat o aplicatie Java care orchestreaza functionarea unui sistem complex de gestiune a livrarii produselor prin intermerdiul unei interfete vizuale intuitive.

Acest proiect este unul pentru uz specific, se poate folosi oricand e nevoie de un sistem de gestiune a unor livrari si efectuarea unor operatii simple de tip CRUD asupra unor date care persista. 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 HashTable, Java Streams, metode de design by contract (post, pre conditii si invariant), sa stapanesc mai bine folosirea design pattern-urilor Composite si Observer 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 interfetei vizuale), mai multe verificari asupra datelor care urmeaza sa fie salvate, mai multe mesaje de eroare sau se pot implementa noi functionalitati precum notificarea clientului cand comanda lui este acceptata de un angajat.

7. Bibliografie

- ✓ https://stackoverflow.com/
- ✓ JavaDoc
  - ➢ https://www.jetbrains.com/help/idea/working-with-code-documentation.html
  - ➢ https://www.baeldung.com/javadoc
- ✓ Adding custom tags to javadoc
  - ➢ https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag
  - ➢ https://maven.apache.org/plugins-archives/maven-javadoc-plugin-2.8.1/index.html
- ✓ Class invariant
  - ➢ https://en.wikipedia.org/wiki/Class_invariant
- ✓ Observer Observable Design Pattern
  - ➢ https://www.baeldung.com/java-observer-pattern
- ✓ Serialization
  - ➢ https://www.baeldung.com/java-serialization
- ✓ WebSockets
  - ➢ https://spring.io/guides/gs/messaging-stomp-websocket/
  - ➢ https://www.baeldung.com/spring-boot-scheduled-websocket
- ✓ Diagrame
  - ➢ https://plantuml.com/sitemap-language-specification
- ✓ Java assert
  - ➢ https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html
- ✓ Lombok
  - ➢ https://projectlombok.org/features/all
- ✓ Thymeleaf
  - ➢ https://stackoverflow.com/questions/25687816/setting-up-a-javascript-variable-from-spring-model-by-using-thymeleaf
- ✓ Bootstrap & FontAwesome (UI)
  - ➢ https://getbootstrap.com/docs/4.0/getting-started/introduction/
  - ➢ https://fontawesome.com/icons/cat-space?style=duotone
- ✓ Range Slider
  - ➢ http://ionden.com/a/plugins/ion.rangeSlider/index.html
- ✓ DataTables
  - ➢ https://datatables.net/
- ✓ Java naming conventions
  - ➢ https://google.github.io/styleguide/javaguide.html