**Documentatie Food Delivery Service**

Pop Mihali Marco Silviu

1. Obiectivul temei

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

3. Proiectare

4. Implementare

6. Concluzii

7. Bibliografie

**1. Obiectivul temei**

Obiectivul principal al temei este de a crea o aplicatie care ajuta la gestionarea unei firme de tip Food Delivery. Aplicatia trebuia sa permita 3 tipuri de utilizatori: admin, angajat si client. Aplicatia trebuie sa permita clientilor sa faca comenzi din lista de produse, adminilor sa modifice/stearga produsele existente si sa adauge produse noi simple si compuse, iar angajatii trebuie sa primeasca notificari de fiecare data cand este creat o comanda.

Obiective secundare:

* Crearea interfetei grafice flutter (dart)
* Preluarea inputului de la utilizator
* Transmiterea datelor catre server
* Prelucrarea datelor (java)
* Serializarea

Crearea interfetei grafice:

In flutter se creeaza o aplicatie web pentru utilizator.

Preluarea inputului de la utilizator:

Datele sunt retinute in front-end inainte de a fi transmise.

Transmiterea datelor catre server

Datele din front-end sunt convertite in obiecte json, dupa care se face un post request catre back-end (java).

Prelucrarea datelor(java)

Datele ajung in back-end si sunt adaugate in structurile de date corespunzatoare. Dupa care se fac verificari suplimentare.

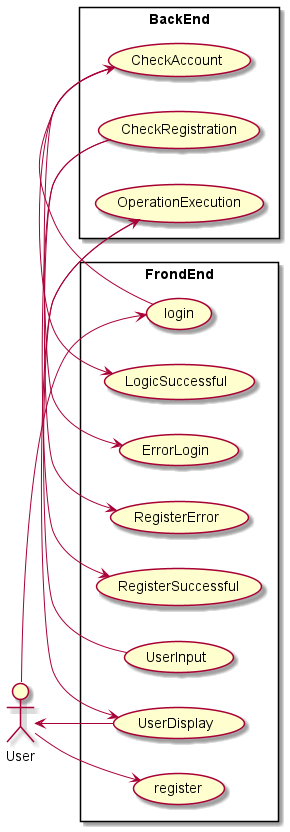
Serializarea

Datele sunt serializare cand se apasa butoanele 'save'/'save and close' din interfata grafica minima din java. Sunt pastrate intr-un fisier din proiect.

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

Cerinte functionale:

* Logare cu cele 3 tipuri de conturi
* Intregistrare
* Ca si client vizualizarea tuturor produselor
* Ca si admin operatii pe produse
* Ca si angajat primirea de notificari cand apare o comanda
* Finalizarea comenzii (client)
* Generarea de raporte (admin)



Use-case diagram

Scenariul principal:

* Utilizatorul introduce numele si parola
* Utilizator este autentificat ca admin, client sau angajat
* Utilizatorul poate efectua diferite operatii
* Datele sunt trimise pe server
* Daca este nevoie clientul primeste un raspuns

Scanariu secundar:

* Utilizatorul introduce numele si parola
* Acestea sunt gresite
* Utilizatorul primeste un dialog box cu eroarea
* Utilizatorul renunta sau se inregistreaza
* Daca numele nu este folosit deja atunci se creeaza contul
* Utilizatorul se poate conecata cu contul deabea creat
* De aici se ajunge in scenariul principal

**3. Proiectare**

Structura, organizarea in pachete:

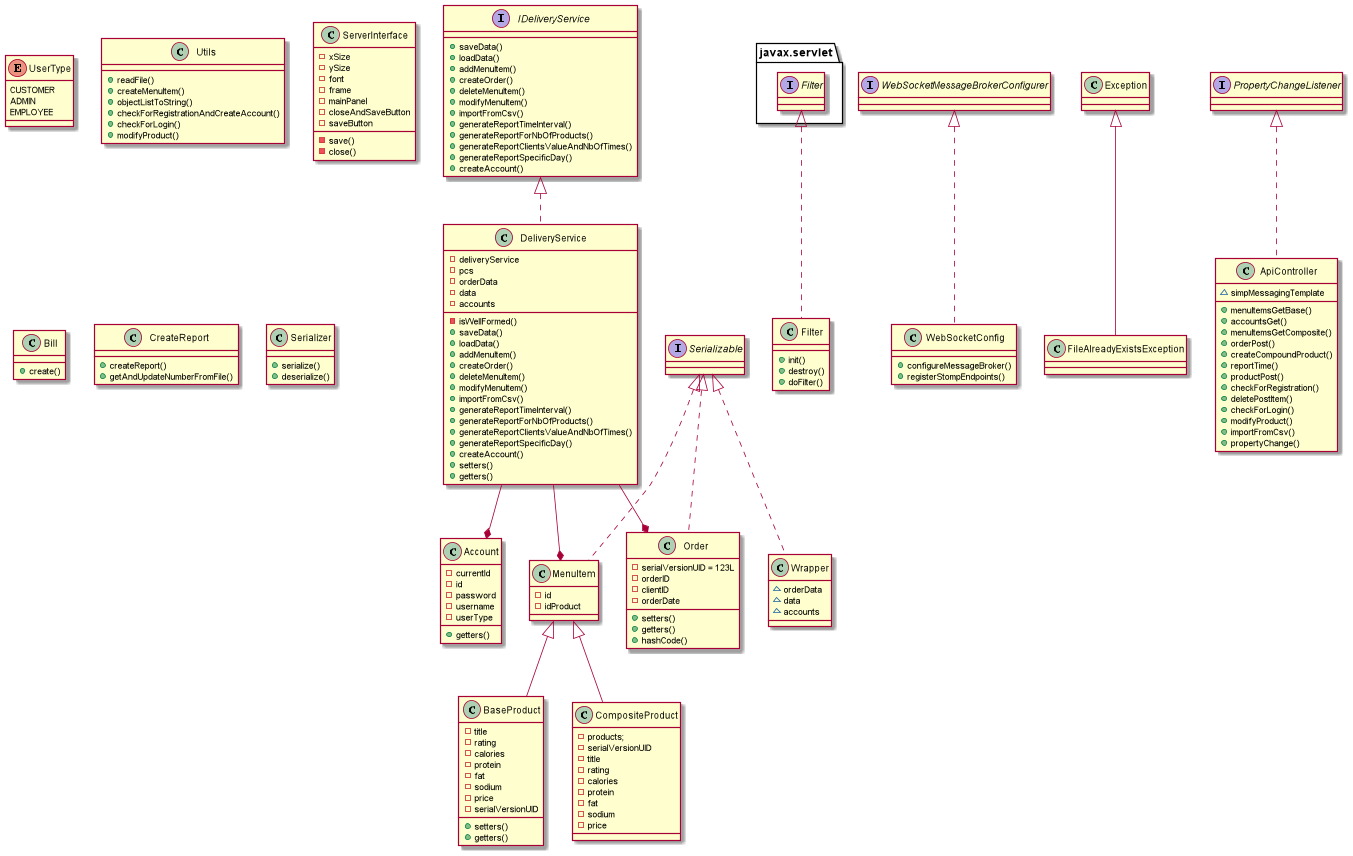
Back-end: business, close.operation, configuration, data, exceptions,

* business : conetine clase pentru retinerea datelor si pentru efectuarea operatiilor pe acestea plus o clasa Utils care ofera multe utilitati necesare
* close.operation: contine o interfata grafica minimala pentru a putea inchise/salva datele prin sincronizare pe server
* configuration: pachet pentru configuratii necesare aplicatiei Spring
* data: contine partea de serializare/deserializare a datelor si si de creare a rapoartelor
* exceptions: contine o exceptie de tip FileAlreadyExistsException
* ApiController: clasa unde sunt definite functii apelate atunci cand se fac request-uri

Front-end: data, pages, server.communication, utils

* data : contine clasele ce retin datele despre conturi, clienti si produse
* pages: contine widget-urile pentru paginile aplicatiei web
* server.communication: contine functiile care fac request-urile catre server si functiile ce stabilesc web socket-urile necesare
* utils: functii si clase utilitare
* Structuri de date folosite:

Pentru retinerea comenzilor am folosit un hash map in care cheia este un obiect de tip Order cu functia de hash suprascrisa, iar ca si valoare o lista de obiecte de tip MenuItem.



UML Class Diagram

**4.1 Implementare Back-end**

Pachetul business:

* Account = retinele datele unui cont : id, parola, nume, tipul utilizatorului
  + Getters and setters
* BaseProduct = retine datele pentru un produs simplu, care nu este compus si anume: titlu, rating, calorii, proteine, grasimi, sare, pret si un serial pt serializare
  + Getters and setters
* CompositeProduct = retine o lista de produse unde caloriile, proteinele, grasimile si sarea sunt calculate ca suma produselor componente, ratingu-ul ca media acestora, iar pretul si numele sunt alese de admin in momentul creeari
* DliveryService = clasa unde se efectueaza operatiile propri-u zise
  + isWellFormed() = metoda ce verifica integritatea datelor din clasa
  + saveData() = metoda pentru serializarea si salvarea datelor curente: conturi, produse, comenzi
  + loadData() = metoda pentru deserializarea datelor si salvarea acestora in sturcturile de date specifice
  + addPropertyChangeListener () = metoda pentru adaugarea unui listener
  + createOrder() = creeaza un order si il adauga in map
  + deleteMenuItem() = sterge un produs din lista de produse
  + modifyMenuItem() = modifica produsul specificat prin id
  + importFromCsv() = aduce in memorie datele din csv-ul dat
  + generateReportTimeInterval() = genereaza un raport cu toate comenzile intre 2 ore indiferent de zi
  + generateReportForNbOfProducts() = genereaza un raport cu produsele ce au fost cumparate de cel putin un anumit nr. de ori
  + generateReportClientsValueAndNbOfTimes() = genereaza un raport cu toate comenzile unde clientul care a facut comanda a comandat de cel putin nb ori, iar valoarea comenzii este cel putin valueMin
  + generateReportSpecificDay() = genereaza raport cu comenzile dintr-o anumita zi
  + createAccount() = creeaza un cont nou si il adauga in lista de conturi
* IDeliveryService = Interfata pentru DeliveryService
* MenuItem = clasa pentru un produs simplu sau compus
  + Getters and setters
  + Un id ce se incrementeaza la fiecare creare de produs nou, indiferent de tipul acestuia
* Order = clasa ce pastreaza datele unei comenzi: id, id-ul clientului, data si ora
  + Getters and setters
* UserType = enumeratie pentru tipurile posibile de utilizatori
  + CUSTOMER
  + ADMIN
  + EMPLOYEE
* Utils = clasa unde se fac citiri si verificari
  + readFile() = citeste din fisierul csv
  + createMenuItem() = dintr-o lista de string-uri creeaza un MenuItem
  + objectListToString() = dintr-o lista de obiecte creeaza un string pentru a fi scris pe factura
  + checkForRegistrationAndCreateAccount() = verifica daca pentru un nume, parola si tip de utilizator exista deja un cont, daca nu il creeaza si il adauga in structura specifica, returneaza o valoarea booleana care semnifica daca a reusit sau nu sa introduca contul nou
  + checkForLogin() = verifica daca un nume si parola exista, in caz pozitiv returneaza obiectul, pentru a stii tipul utilizatorului logat
  + modifyProduct() = modifica un produs

Pachetul close.operation:

* ServerInterface = pachet pentru interfata grafica minimala a serverului, pentru a nu serializa datele la fiecare mica schimbre si a ingreuna programul, am creat o interfata grafica cu 2 butoare.
  + Butonul Save = serializeaza datele din DeliveryService dupa care le scrie intr-un fisier
  + Buton Save and Exit = serializeaza datele din DeliveryService dupa care le scrie intr-un fisier dupa care inchide aplicatia

Pachetul configuration:

* Filter = ajuta la configurarea aplicatiei Spring si la headerele trimise
* WebSocketConfig = clasa ce ajuta la stabilirea web socketului ce notifica angajatul de crearea unei noi comenzi

Pachetul data:

* Bill = creaza factura unei comenzi
  + create() = primeste o comanda, si lista de comenzi
* CreateReport = creeaza rapoarte
  + createReport () = creeaza fisierul pentru raport si dupa scrie in el lista de obiecte primite ca string
  + getAndUpdateNumberFromFile () = foloseste un id care este intr-un fisier pentru ca id-ul sa fie persistent si intre
* Serializer = clasa ce serializeaza/deserializeaza date
  + serialize() = primeste un obiect de tip Serializable si il scrie in fisierul de tip out folosind streams
  + deserialize() = returneaza un obiect din fisierul unde au fost salvate datele serializate
* Wrapper() = clasa ce infasoara lista de produse, lista de conturi si map-ul cu comenzi pentru a crea un singur obiect serializabil

Pachetul exceptions:

* FileAlreadyExistsException() = exceptie aruncata la crearea unui raport atunci cand fisierul ce se doreste a fi creat exista deja

Clasa ApiController : aici sunt toate functiile apelare atunci cand un request este facut de client

* menuItemsGetBase() = metoda ce trimite lista de produse de baza
* accountsGet() = metoda ce trimite lista de conturi existente
* menuItemsGetComposite()metoda ce trimite lista de produse compuse
* orderPost() = functie care este chemata atunci cand apare un request de tip post cu o comanda, json-ul primit este decodat intr-un obiect de tip Order dupa care adaugat in map
* createCompoundProduct() = primeste o lista de produse, un titlu si un pret de la client, date din care creeaza un nou produs compus pe care il adauga la lista
* reportTime() = atunci cand utilizatorul de tip admin doreste sa creeze un raport cu toate comanzile intre 2 ore el alege cele 2 ore din interfata, dupa care se face un request de tip post si datele ajungi in aceasta functie, dupa care se genereaza raportul
* productPost() = functie chemata atunci cand utilizatorul de tip admin doreste sa creeze un nou produs de baza
* checkForRegistration() = atunci cand se doreste inregistrarea unui nou cont aceasta functie verifica daca acesta exista deja sau nu si returneaza o valoare bool
* deletePostItem() = primeste numele unui produs pe care il sterge din lista da produse
* checkForLogin() = verifica daca numele si parola au fost inregistrare, daca da returneaza un obiect Account pentru ca pe front-end sa se stie ce fel de tip are utilizatorul
* modifyProduct() = primeste un produs vechi si unui nou, inlocuieste valorile produsului vechi cu cele ale produsului nou in lista de produse
* importFromCsv() = atunci cand adminul doreste importarea datelor din csv
* propertyChange() = atunci cand se creeaza un nou order se trimite aceste printr-un web socket

**4.2 Implementare Front-end**

Director data:

* Account = clasa ce retine datele unui cont, si se ocupa de codificare si decodificare json
* CompositeMenuItem = clasa ce retine datele unui produs compus si codificare decodificare json
* MenuItem= clasa ce retine datele unui produs compus si codificare decodificare json
* Order = clasa ce retine datele unei comenzi si codificare decodificare json
* UserType = enumeratie cu tipurile de cont : ADMIN, CUSTOMER, EMPLOYEE

Director Pages:

* Subdirector operation.pages :
  + AddCompoundProductPage = wigdet pentru pagina ce permite adminului sa adauge un produs compus
  + AddProductPage= wigdet pentru pagina ce permite adminului sa adauge un produs
  + DeleteModifyProductPage = wigdet pentru pagina ce permite adminului sa stearga un produs din cele existente
  + Reports = widget ce permite adminului sa selecteze 2 ore pentru crearea raportului cu toate comenzile dintre orele alese
  + ViewCompoundProductPage = widget ce permite adminului sa selecteze mai multe produse pentru a crea un produs compus
* AdminPage = widget pentru pagina adminului ce permite alegerea operatiei dorite
* ClientPage = permite cautarea produselor, adaugarea lor la o comanda
* EmployeePage = widget pentru pagina angajatiilor unde apar comenzile facute
* OrderPage = widget pentru pagina unde se finalizeaza comanda
* RegisterPage = widget pentru pagina unde un utilizator se poate inregistra
* StartPage = widget pentru pagina de logare

Director server.communicaton:

* dataService:
  + fetchDataBase = request get pentru lista de BaseProduct
  + fetchDataComposite = request get pentru lista de BaseProduct
  + postOrderData = request post cand se introduce o comanda
  + postBaseProduct = request post cand se adauga un produs de baza
  + deletePostItem = request post pentru a sterge un produs din lista
  + modifyMenuItem = request post pentru a modifica un produs din lista
  + createCompoundProduct = request post la o lista de produse un titlu si un pret pentru a fi creat in server
  + importFromCsv = request get pentru a importa in server din csv
  + reportGenerate = request post pentru generarea raportului de timp
  + checkIfRegistered = request post pentru a afla daca inregistrarea e valida
  + checkIfHasAccount = request post pentru a afla daca utilizatoul se poate loga
* sockets: stomp sockets functions

Director utils:

* Constraints
* TempProductCreate
* utils

Interfata grafica:

* Pagina de logare, aici utilizatorul isi introduce datele petru a se loga, in cazul in care este corect acesta va fi redirectioant la pagina de admin, client sau angajat, in cazul in care nu apare o eroare
* Pagina clientului unde clientului poate alege sa adauge produse in order, dupa care se poate muta pe pagina de ViewOrder unde poate termina comanda
* Pagina angajatului, unde angajatul primeste notificari referitoare la comenzi
* Pagind administratorului unde se poate alege dintre : Add Product, Delete/Modify product, Create new compound product, Import from csv, Create reports

**6. Concluzii**

Am facut o aplicatie de tip DeliveryFood pentru clienti, angajati si administratori in care administratorii pot modifica sterge sau adauga produse, clientii pot face comenzi din mai multe produse, iar angajatii pot primi notificari si alege comenzile pe care le pot livra. S-ar putea aduce imbunatatiri la partea grafica, sa se uniformizeze modelele. La partea java s-ar putea face mai multe verificari pentru a asigura validitatea datelor si usoarea detectare a erorilor.

**7. Bibliografie**

* stackoverflow.com
* www.baeldung.com
* https://pub.dev/packages/hovering/example
* https://flutter.dev
* https://www.w3schools.com