Gestionarea datelor unui restaurant

Faliboga Dimitrie

December 31, 2024

Laborator 1

Output

Output-ul este sub forma de mesaje in consola.

```
if (authSystem.isAdmin(currentUser.getUsername())) {
    System.out.println("Successfully logged in as ADMIN!");
    adminAuthOptions(authSystem,scanner);
} else {
    System.out.println("Only ADMIN can register STAFF or other ADMIN users!");
    return false;
}
```

Tipuri de valori

Programul foloseste mai mult tipuri de valori: primitive, de timp, generice.

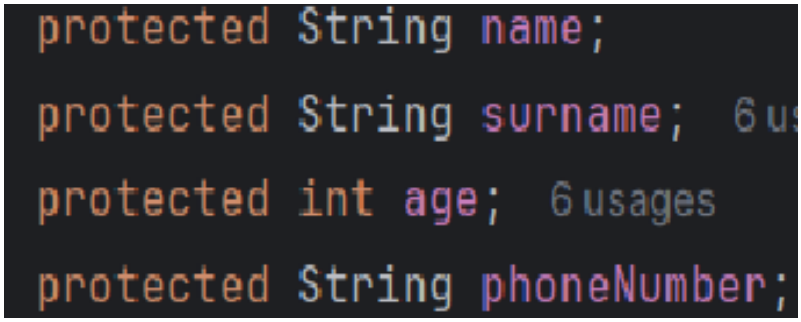

Figure 1: Exemple de tipuri primitive

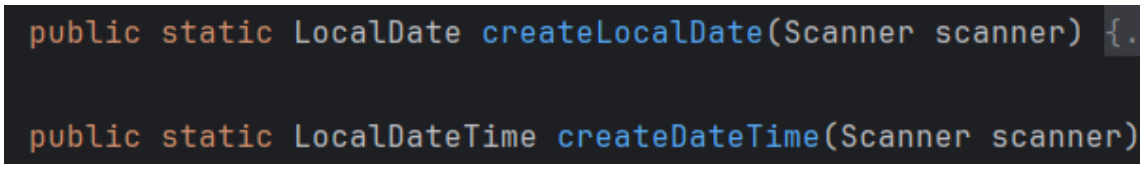

Figure 2: Metode ce returneaza tipuri de timp

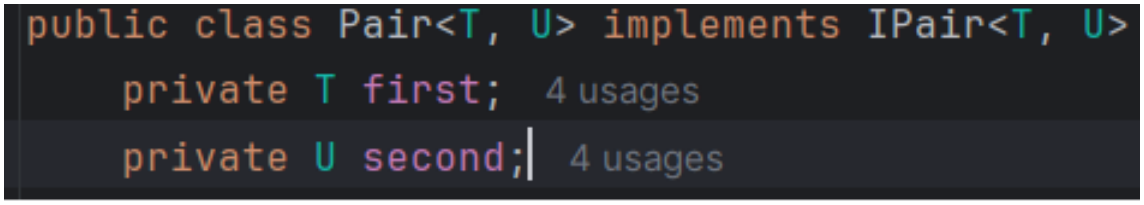

Figure 3: Implementare Pair ce foloseste 2 tipuri de date primitive

Functii

Main utilizeaza diferite functii, pentru a pastra codul modular.

```
boolean continueApp = false;
loadDataFromJson(authSystem, restaurant);
while (!continueApp) {
    continueApp =  beginConsoleIO(authSystem);
}

if (authSystem.getCurrentUser() != null) {
    mainConsoleIO(authSystem.getCurrentUser().getRole(), restaurant);
}
```

Figure 4: Cele 3 functii principale, singurele folosite direct in metoda main

Laborator 2

Input

Inputul este citit de la tastatura, cu ajutorul scanner-ului.

```
System.out.println("Please input a value of type " + type.getSimpleName() + ":");
String input = scanner.nextLine();
```

Figure 5: Datele sunt citite intr-un string, apoi interpretate

Structuri repetitive si conditionale

Programul utilizeaza mai multe astfel de structuri. In imaginea 4 se exemplifica utilizarea structurii repetitive while, urmata de structura conditionala if. Sunt utilizate si structuri repetitive de tip for, precum si structura conditionala switch.

```
System.out.println("Please choose an option:");
int index = 1;
for (E option : enumClass.getEnumConstants()) {
    System.out.println(index + ". " + option);
    index++;
}
```

Figure 6: Exemplu de for each, ce poate parcurge orice enum

```
switch (type) {
    case DRINK -> {
        return findDrinkByName(name);
    }
    case FOOD -> {
        return findFoodByName(name);
    }
    default -> throw new UnsupportedOperationException("Unsupported menu item type: " + type);

}
```

Figure 7: Exemplu de patter matching folosind switch

Laborator 3

In program am utilizat, in principal, 2 structuri de date:

1. ArrayList - din care am folosit metode precum: .add(element) care adauga la coada vectorului elementul; .remove(String) - cauta string-ul in vector si il elimina
2. HashMap - cele mai utilizate metode: put(cheie, valoare); containsKey(cheie) - verifica daca exista cheia respectiva in hash table; .entrySet() - utilizat intr-un range base for pentru a itera prin hashMap.

```
for (Map.Entry<Integer, Integer> entry : tablesByNumber.entrySet()) {
    if (entry.getValue() > tableIndex) {
        tablesByNumber.put(entry.getKey(), entry.getValue() - 1);
    }
}
```

Figure 8: Exemplu de .entrySet()

Laborator 4

Am utilizat mai mult clase (vezi diagrama UML a claselor 11), asa ca o sa iau o clasa mai mica, ca sa o prezint.

```
public class User implements IUser {
//atribute
private String username;
private String password;
private Role role;

//Metode
public User() // constructor fara parametri
public User(String username, String password, Role role) // constructor cu parametrii
public String toString() //Intoarce un string care contine username si role
//Getter si Setter pentru cele 3 atribute
}
```

Laborator 5

Ca si in cazul laboratorului 4, se pot observa 2 mosteniri in UML-ul claselor.

- Employee mosteneste Person - o mostenire evidenta, angajatul (consideram ca numai oamenii pot fi angajati) este o persoana, cu anumite atribute in plus.
- Clasele Drink,Food mostenesc MenuItem - in contextul unui meniu, este adevarat deoarece ambele au elemente comune precum: nume, calorii, pret etc. deci ambele sunt de tipul menuItem, extinzandu-l cu caracteristici proprii (Drink are volum, Food are tip de cuisine).

Laborator 6

Am facut interfete pentru anumite clase, cele mai utilizate, in care am doar metode ce ar trebuii sa existe indiferent de cum se comporta clasa ce implementeaza interfata. Ex: Table are interfata Itable care contine 3 metode - reserveTable(), occupyTable() si freeTable() - care sunt general valabile in orice Restaurant, si in marea majoritate a cazurilor inafara restaurantului. Vezi diagrama interfetelor 12, 13.

Laborator 7

Din cauza dimensiunii aplicatiei, am facut teste automate numai pentru cateva metode (getter, setter si toString sunt banale deci nu le-am luat in considerare).

```java
@org.junit.jupiter.api.Test  Dimitrie Faliboga
void authenticate() {
    AuthSystem authSystem = new AuthSystem();
    User user = new User( username: "test",  password: "test", Role.ADMIN);
    authSystem.addUser(user);

    User authenticatedUser = authSystem.authenticate( username: "test",  password: "test");
    assertNotNull(authenticatedUser);
    assertEquals((user.getUsername()), authenticatedUser.getUsername());

    authenticatedUser = authSystem.authenticate( username: "testUser",  password: "wrongPass");
    assertNull(authenticatedUser);

    authenticatedUser = authSystem.authenticate( username: "nonexistentUser",  password: "anyPass");
    assertNull(authenticatedUser);
}
```

Figure 9: Testeaza autentificare unui utilizator - diagrama 14

```java
@org.junit.jupiter.api.Test  Dimitrie Faliboga
void register() {

    AuthSystem authSystem = new AuthSystem();
    authSystem.register( username: "test",  password: "test", Role.ADMIN);
    User user = authSystem.getUsers().get("test");
    assertNotNull(user);
    assertEquals( expected: "test", user.getUsername());
    assertEquals( expected: "test", user.getPassword());
    assertEquals(Role.ADMIN, user.getRole());

    assertThrows(IllegalArgumentException.class, () -> authSystem.register( username: "test",  password: "test", Role.ADMIN));
}
```

Figure 10: Testeaza inregistrarea unui utilizator - diagrama 15

Laborator 8

Am folosit fisiere de tip Json pentru a salva datele. Serializarea si deserializarea datelor este gestionata de JsonUtil. Acesta are urmatoarele metode, toate generice:

1. `saveToJson(List<T> items, String fileName)` -

   sterge continutul fisierului si incarca elementele din lista items

2. `sappendToJson(List<T> items, String fileName, Class<T> clazz)` -

   adauga la finalul fisierului elementele din lista items, cere sa fie specificata clasa obiectelor din lista.

3. `appendToJson(T item, String fileName, Class<T> clazz)` -

   la fel ca metoda anterioara, numai ca accepta un singur element

4. `List<T> loadFromJson(String fileName, Class<T> clazz)` -

   intoarce o lista cu toate elemente din acel fisier

5. `void removeFromJson(String fileName, Class<T> clazz, Predicate<T> condition)` -

   elimina toate elementele din Json care respecta conditia data

6. ```
   void updateElementInJson(
   String fileName,
   Class<T> clazz,
   Predicate<T> condition,
   Consumer<T> updater) -
   ```

   metoda care parcurge tot fisierul, selecteaza elementul care respect conditia, si, daca se poate face modificarea lui via updater, acesta se modifica.

Incarcarea datelor din fisierele Json se poate observa in imaginea 4 - prin metoda loadDataFromJson Salvarea datelor se face in fisiere diferite pentru fiecare tip de obiect.
   ```

Laborator 9

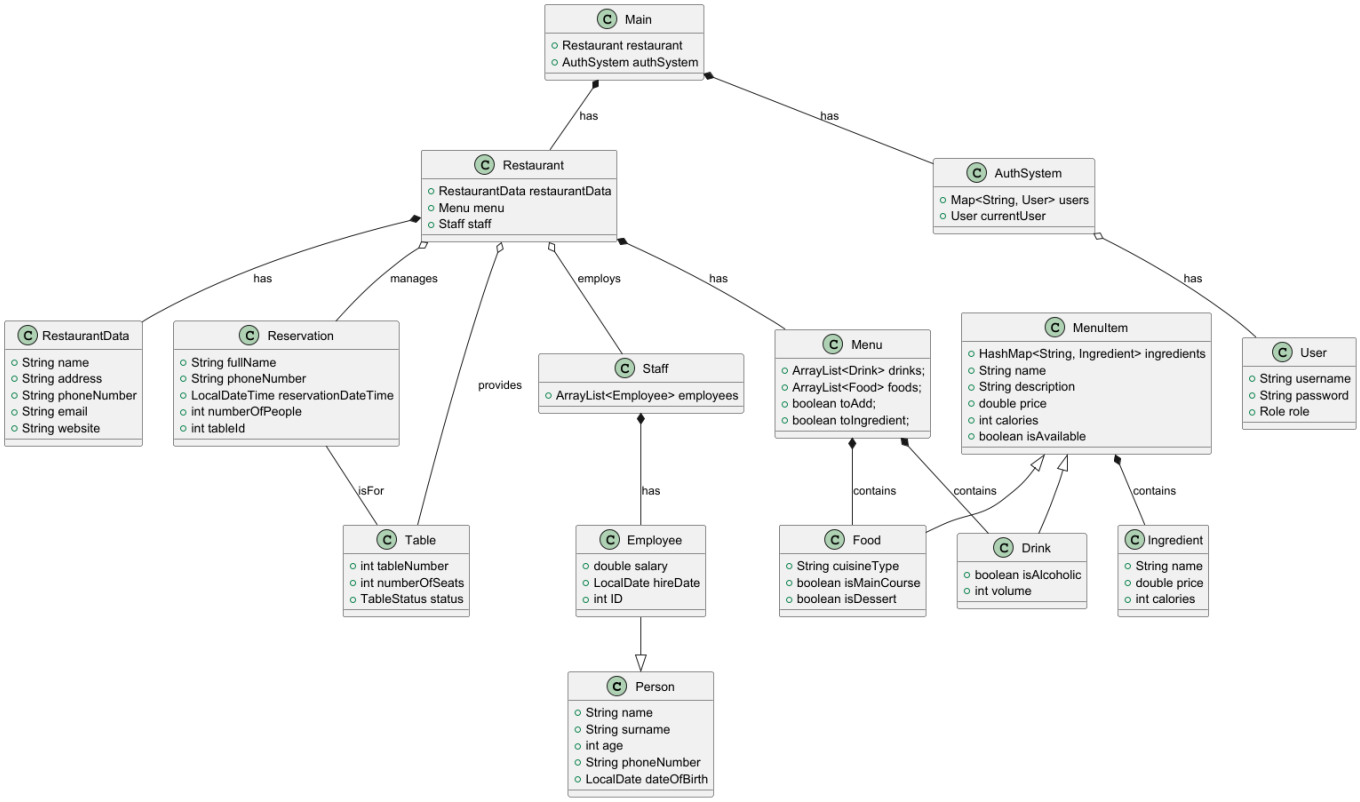

Figure 11: Diagrama claselor

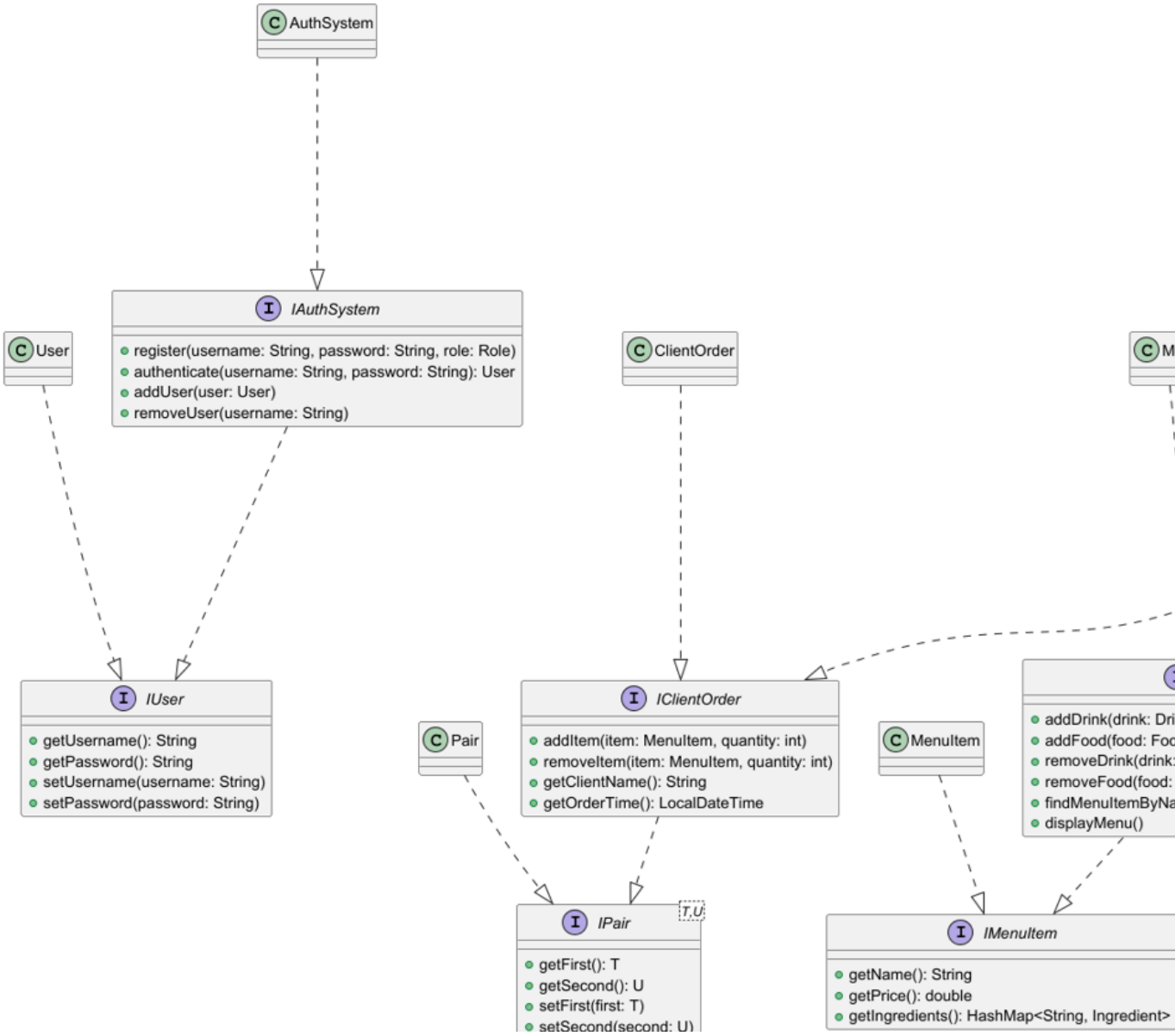

Figure 12: Diagrama interfetelor - a doua jumatate

Figure 13: Diagrama interfetelor - prima jumatate

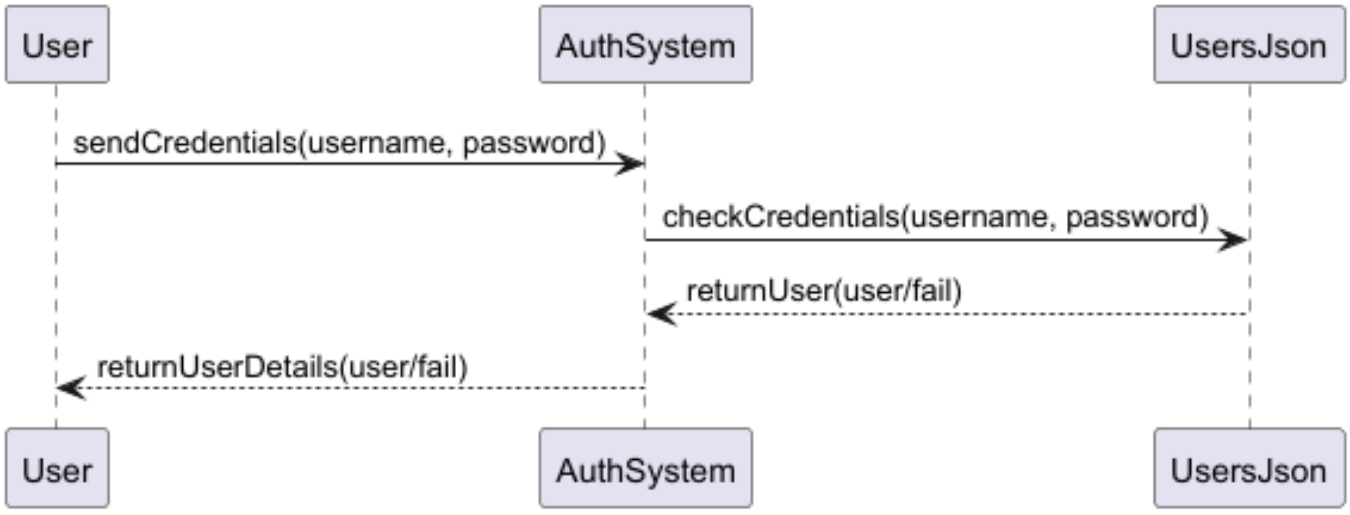

Figure 14: Diagrama logare user

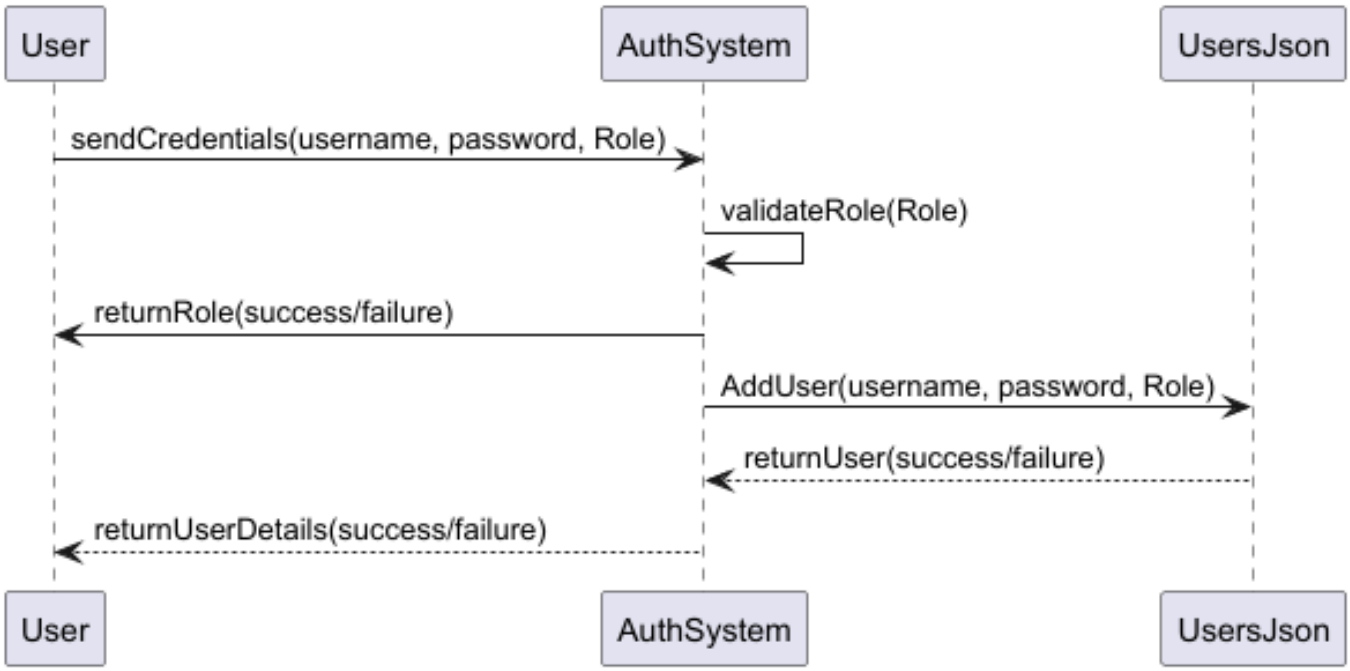

Figure 15: diagrama inregistrare user