Student: IEREMIAȘ Viorel

Grupa: 244

Tema 2 – Middleware

Tematică: Agenda evenimente

# Cerință

*“Agenda: Intretinerea unei DB cu evenimente:*

*\* Adaugare / stergere / modificare eveniment*

*\* Lista cronologica a evenimentelor*

*\* Lista evenimente filtrata dupa parti din continut*

*Problema: Agenda, tehnologia: Pyro.”*

# Soluție propusă

## Server

In etapa de design a modelului de date se identifica nevoia unei singure entitati pentru reprezentarea evenimentelor. In implementare, aceasta se numeste **CalendarEvent** si are urmatoare campuri:

* name – sir de caractere cu lungimea maxima de 100
* description – sir de caractere cu lungimea maxima de 500
* scheduled\_time – structura de tip **DateTime**

Serverul este implementat conform cerintei in Python 3. Persistenta datelor este realizata folosind o baza de date **MySQL.** Ca ORM se foloseste **sqlalchemy**, cel mai popular si folosit ORM pentru Python.

Acesta permite definirea campurilor unei clase care urmeaza sa fie persistate prin extinderea unui **Base** class si folosirea tipurilor predefinite (**Column**, **Integer**, **String**, **DateTime**, etc.). Operatiile **CRUD** standard sunt disponibile in clasa **Session** fara a fi necesara scrierea explicita de query-uri SQL. Ordonarea sau filtrarea rezultatelor se face de asemenea programatic, prin metode puse la dispozitie in clasa **Session** (**order**, **filter**). Crearea sesiunilor se face cu ajutorul unui Session Factory creat in momenul in care se seteaza atributele bazei de date cu care se comunica.

Initializarea conexiunii la baza de date presupune definirea unui sir de caractere care contine dialectul, credetialele utilizatorului, URL-ul serverului de baze de date si numele aceasteia.

Pentru a separa stratul de persistenta de cel de prezentare si din motive de siguranta, exista entitatea **CalendarEvent**, care defineste inclusiv informatiile necesare pentru ORM, si **CalendarEventDTO**, clasa care este folosita in comunicarea cu clientii. Maparea intre cele 2 este facuta de 2 metode statice din clasa **CalendarEventDTO**.

La nivelul de servicii, exista un **CalendarEventService** care defineste metodele expuse spre clienti. Implementarea este delegata spre nivelul de persistenta (clasa **CalendarEvent** in principal), insa la acest nivel se face translatarea intre **CalendarEvent** si **CalendarEventDTO** si invers. **ExecPyro4Serv** este serverul **Pyro4** care expune functiile puse la dispozitie de **CalendarEventService** plus alte servicii daca este necesar.

Clasa principala, **Main** realizeaza pasii de initializare a sistemului:

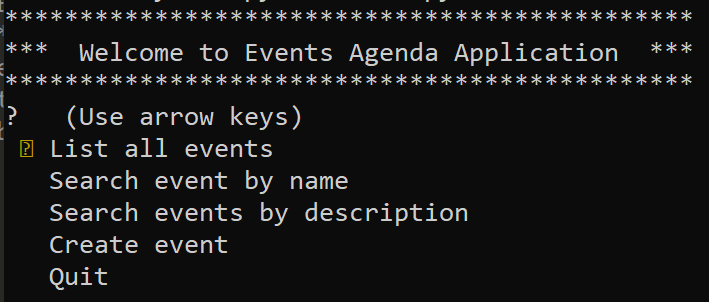
* creeaza conexiunea la baza de date
* creaza **SessionFactory**-ul folosit de stratul de persistenta
* porneste serverul **Pyro**

## Clienti

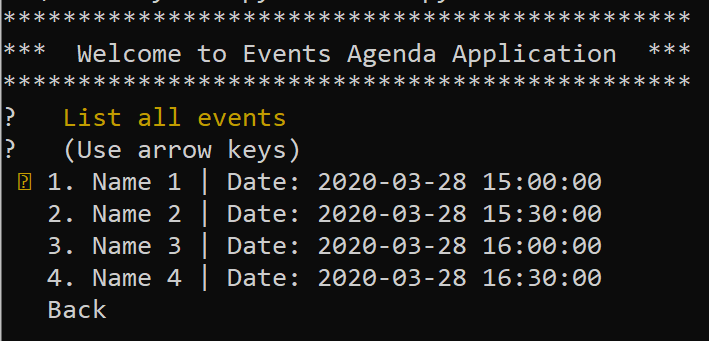
S-au implementat 3 clienti pentru serverul descris mai sus, in Python, Java si C#. Clientii Java si C# folosesc pachate **Pyrolite** specifce pentru a se conecta la serverul **Pyro**. Toti cei 3 clienti sunt aplicatii **CLI** (command-line interface) **interactive** (permit navigarea cu ajutorul sagetilor). In plus, exceptand particularitatile de sintaxa ale limbajului in care sunt dezvoltati, cei 3 clienti sunt de asemenea aproape identici.

In pachetul **model** fiecare contine specificatia clasei **CalendarEventDTO** si validatori pentru numele unui eveniment si pentru formatul campului **scheduled\_time**. Pachetul **cli** contine clase **enum** care contin optiunile afisate utilizatorului in diferite situatii in aplicatie. De asemenea, sunt definite metodele care proceseaza alegerile utilizatorului.

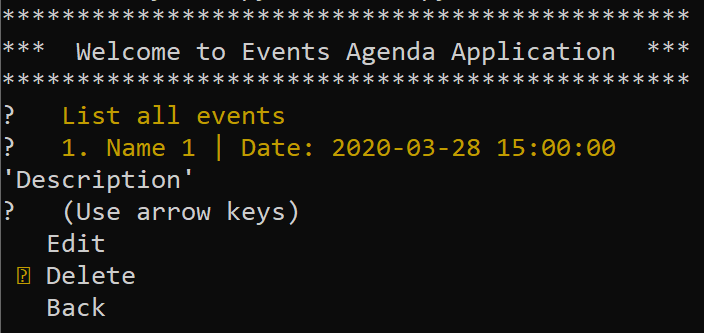
Implementarea interfetei interactive se bazeaza pentru fiecare client pe cate un pachet care ofera urmatoarea functionalitate: afisarea unei liste de optiuni, navigarea cu ajutorul sagetilor pentru selectarea unui optiuni, returnarea raspunsului. Pentru clientul Python acest pachet este **PyInquirer**, pentru Java este **Text-IO**, iar pentru C# este **EasyConsole**.

 Clasa **Main** creeaza clientul **Pyro** si initializeaza interfata.

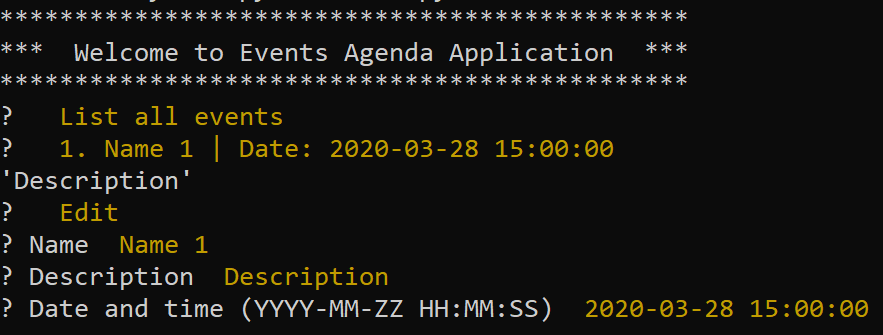
*Meniul principal*



*Lista evenimentelor ordonate cronologic*

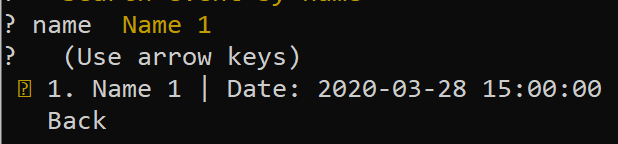


*Operatiunile disponibile pentru un eveniment*



*Formularul de editare a unui eveniment, cu valorile existente precompletate*

*(formularul de creare este identic, insa nu vine precompletat cu nicio valoare)*



*Cautarea unui eveniment dupa nume*

*(cautarea dupa cuvinte cheie este asemanatoare, insa nu la fel de stricta)*

# Instalare și rulare

1. Server

* Se ruleaza comanda pip install -r requirements.txt
* Se ruleaza comanda python main.py

1. Client Python

* Se ruleaza comanda pip install -r requirements.txt
* Se ruleaza comanda python main.py

1. Client Java

* Se ruleză comand gradle clean build
* Se lansează în execuție fișierul JAR autolansabil rezultat din fodlerul build/libs folosind comanda java -jar ClientJava.jar

1. Client C#

* Se deschide soluția ClientCS.sln folosind mediul de dezvoltare Visual Studio
* Se ruleaza opțiunea Build Solution din meniul Build
* Se lansează în execuție fișierul executabil rezultat din folderul ClientCS \bin\Debug\netcoreapp3.1 folosind comanda .\ ClientCS.exe