Особливості розробки алгоритмів методів програмних класів

DAO (Data Access Object) - це шар об'єктів які забезпечують доступ до даних.

Зазвичай для реалізації DAO використовується EntityManager і з його допомогою ми працюємо з нашою БД, але в нашому випадку це система не підійде, так як ми вивчаємо Spring Data нам потрібно використовувати її засоби інакше нема чого він нам.

Spring Data надає набір готових реалізацій для створення DAO але Spring вважали за краще цей шар називати не DAO, а Repository.

Приклад створення Repository:

package pl.sda.finalProject.myOrganizer.dao;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import pl.sda.finalProject.myOrganizer.entity.Event;

import pl.sda.finalProject.myOrganizer.entity.MyUser;

import java.util.List;

@Repository

public interface IEventRepository extends JpaRepository<Event, Long> {

List<Event> findByUserOrderByEventDateAsc(MyUser user);

List<Event> findAllByUser(MyUser user);

void deleteAllByUser(MyUser user);

}

JpaRepository - це інтерфейс фреймворка Spring Data надає набір стандартних методів JPA для роботи з БД.

Service - це Java клас, який надає з себе основну (Бізнес-Логіку). В основному сервіс використовує готові DAO / Repositories або ж інші сервіси, для того щоб надати кінцеві дані для призначеного для користувача інтерфейсу.

Приклад створення Service:

package pl.sda.finalProject.myOrganizer.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import pl.sda.finalProject.myOrganizer.dao.IEventRepository;

import pl.sda.finalProject.myOrganizer.entity.Event;

import pl.sda.finalProject.myOrganizer.entity.MyUser;

import pl.sda.finalProject.myOrganizer.model.EventModel;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

import java.util.Iterator;

import java.util.List;

import java.util.stream.Collectors;

@Service

public class EventService {

@Autowired

private IEventRepository eventRepository;

@Autowired

private DateTimeFormatter dateFormatter;

@Autowired

private DateTimeFormatter timeFormatter;

public void parseEventDateAndTime(EventModel eventModel) {

// eventDate required

Продовження прикладу створення Service:

eventModel.setEventDate(LocalDate.*parse*(eventModel.getStringEventDate(), dateFormatter));

// optional

if (!eventModel.getStringEventTime().isEmpty()) {

eventModel.setEventTime(LocalTime.*parse*(eventModel.getStringEventTime(), timeFormatter));

}

}

public String parseEventDate(LocalDate localDate) {

return localDate.format(dateFormatter);

}

public String parseEventTime(LocalTime localTime) {

return localTime.format(timeFormatter);

}

public List<Event> getCurrentEvents(MyUser activeUser) {

List<Event> events = eventRepository.findByUserOrderByEventDateAsc(activeUser);

for (Iterator<Event> iterator = events.iterator(); iterator.hasNext(); ) {

Event event = iterator.next();

if (event.getEventDate().isBefore(LocalDate.*now*())) {

eventRepository.delete(event);

iterator.remove();

}

}

return events;

}

public List<Event> getEventsToRemind(MyUser activeUser) {

return eventRepository.findAllByUser(activeUser).stream().filter(

event -> isEventToRemind(event)

).collect(Collectors.*toList*());

}

private boolean isEventToRemind(Event event) {

if (event.getEventTime() == null) {

event.setEventTime(LocalTime.*MIDNIGHT*);

}

if (event.getDaysBefore() > 0 && event.getEventDate().

minusDays(event.getDaysBefore()).isEqual(LocalDate.*now*())) {

return true;

}

if (event.getHoursBefore() > 0 && LocalDateTime.*now*().isAfter(LocalDateTime.*of*(event.getEventDate(),

event.getEventTime()).minusHours(event.getHoursBefore()))) {

return true;

}

if (event.getMinutesBefore() > 0 && LocalDateTime.*now*().isAfter(LocalDateTime.*of*(event.getEventDate(),

event.getEventTime()).minusMinutes(event.getMinutesBefore()))) {

return true;

} else return false;

}

}