Programare Paralelă și Distribuită

Proiect 1

Haja Florin Gabriel, Gabor Vlad Sebastian, grupa 233/2

**Proiect 1 – Client-Server**

Obiectiv:

- Folosirea executiei concurente prin apeluri asincrone.

- Folosirea mecanismelor: future/promises si thread\_pool.

- Analiza imbunatatirii performantei executiei unei aplicatii (de tip business) prin

programare concurenta.

**Sala concerte**

O sala de concerte vinde bilete la spectacolele organizate printr-o aplicatie client-server.

Sala organizeaza cel mult un spectacol pe zi.

Sala de concerte are un numar maxim - ‘nr\_locuri’ - de locuri numerotate de la 1 la ‘nr\_locuri’.

Pentru fiecare spectacol avem informatii de tip (data, titlu).

Permanent sala mentine o evidenta actualizata pentru:

- informatii despre bilete pentru fiecare spectacol - (ID\_spectacol, lista\_locuri\_vandute);

- vanzarile efectuate: lista de vanzari; vanzare = (data\_vanzare, ID\_spectacol, numar\_bilete,

lista\_locurilor) ;

- soldul total per spectacol(suma totala incasata).

Periodic sistemul (2 cazuri testare: 5, 10 secunde) face o verificare a locurilor vandute prin verificarea corespondentei corecte intre locurile libere si vanzarile facute, sumele incasate per vanzare si soldul total.

Sistemul foloseste un mecanism de tip ‘Thread-Pool’ pentru rezolvarea a taskurilor.

Pentru a testare se va considera ca fiecare client initiaza/creeaza la interval de 2 sec o noua cerere de vanzare

bilete folosind date generate aleatoriu (nr\_de\_bilete, locuri) si se primeste de la server o notificare – vanzare reusita sau vanzare nereusita. Nu este necesara interfata grafica!

Pentru verificare se cere salvarea pe suport extern (fisier text, sau BD) a vanzarilor si a rezultatelor operatiilor de verificare executate periodic: data, ora, sold\_per spectacol, lista vanzarilor per spectacol, ‘corect/incorect’.

Serverul se inchide dupa un interval de timp precizat si notifica clientii activi referitor la inchidere.

Optional (pentru 2 puncte suplimentare – actualizare la nivel client al listei locurilor ocupate - Observer).

Model

Spectacol (ID\_spectacol, data\_spectacol, titlu, pret\_bilet, lista\_locuri\_vandute, sold)

Vanzare (ID\_spectacol, data\_vanzare, nr\_bilete\_vandute, lista\_locuri\_vandute, suma)

Sala(nr\_locuri, Lista<Spectacol>, Lista<Vanzari>)

Taskuri posibile

* Vanzare bilete
* Verificare

Limbajul de implementare: la alegere

**Testare:**

Nr\_locuri =100; / 500  
3 spectacole (S1, S2, S3)

S1 pret\_bilet=100;

S2 pret\_bilet=200;

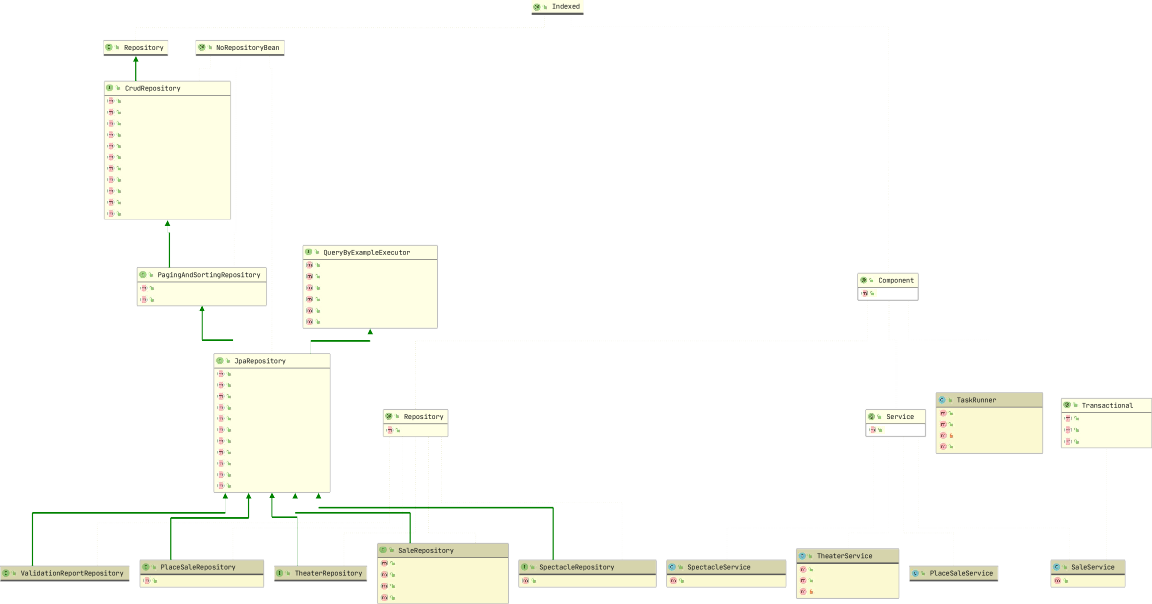
S3 pret\_bilet=150;

Serverul lucreaza 2 minute.

Verificare

-Data, ora, corect/incorect --- per spectacol

Diagrama claselor arată astfel:



getTakenPlaces(long) List<Integer>

add(SaleDto) long

getNumberOfPlaces(long) int checkDataValidity() void

seekRepositories() void

getAllBySaleId(long) List<PlaceSale>

getAllBySpectacleId(long) List<Sale> getTakenPlacesById(long) List<Integer> existsPlaceForSpectacleId(int, long) int countPlacesForSpectacleId(long) int

findAllByTheater(Theater) List<Spectacle>

value() String

value() String

value() TxType

rollbackOn() Class[] dontRollbackOn() Class[]

builder(TaskType) Builder

initialize() void

addInitialTasks() void addTask(Task) Future<Object>

findAllById(Iterable<ID>) List<T> saveAll(Iterable<S>) List<S>

flush() void

saveAndFlush(S) S deleteInBatch(Iterable<T>) void deleteAllInBatch() void

getOne(ID) T

findAll(Example<S>) List<S> findAll(Example<S>, Sort) List<S>

List<T> List<T>

findAll() findAll(Sort)

value() String

findOne(Example<S>) Optional<S> findAll(Example<S>) Iterable<S> findAll(Example<S>, Sort) Iterable<S> findAll(Example<S>, Pageable) Page<S> count(Example<S>) long

exists(Example<S>) boolean

findAll(Sort) Iterable<T>

findAll(Pageable) Page<T>

saveAll(Iterable<S>) Iterable<S> findById(ID) Optional<T>

existsById(ID) boolean

findAll() Iterable<T> findAllById(Iterable<ID>) Iterable<T> count() long

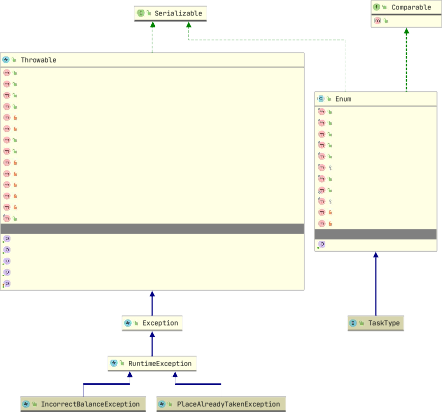
deleteById(ID) void

delete(T) void

deleteAll(Iterable<? extends T>) void deleteAll() void

S

save(S)



name() String

ordinal() int

toString() String

equals(Object) boolean

hashCode() int

clone() Object

compareTo(E) int

valueOf(Class<T>, String) T

finalize() void

readObject(ObjectInputStream) void readObjectNoData() void

declaringClass

Class<E>

Throwable String Throwable[] String

StackTraceElement[]

cause message suppressed

localizedMessage stackTrace

printEnclosedStackTrace(PrintStreamOrWriter, StackTraceElement[], String, String, Set<Throwable>) void printStackTrace(PrintWriter) void

fillInStackTrace() Throwable

fillInStackTrace(int) Throwable

getOurStackTrace() StackTraceElement[]

readObject(ObjectInputStream) void

validateSuppressedExceptionsList(List<Throwable>) int

writeObject(ObjectOutputStream) void

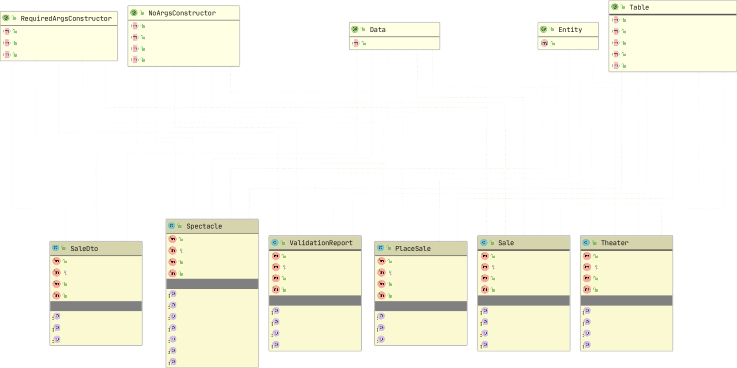
addSuppressed(Throwable) void

Throwable String void void void

initCause(Throwable) toString() printStackTrace()

printStackTrace(PrintStream) printStackTrace(PrintStreamOrWriter)

compareTo(T) int



*equals(Object)* boolean

*canEqual(Object)* boolean

*equals(Object)* boolean *canEqual(Object)* boolean *hashCode()* int

*toString()* String

*hashCode() toString()*

int String

*equals(Object)* boolean

*canEqual(Object)* boolean

*equals(Object)* boolean *canEqual(Object)* boolean *hashCode()* int

*sellingDate* LocalDate *placeSales* List<Integer> *spectacleId* long

*title theater balance sales data*

*ticketPrice id*

String Theater long List<Sale> LocalDate long

long

*hashCode() toString()*

int String

*equals(Object)* boolean

*canEqual(Object)* boolean

*hashCode() toString()*

int String

*toString()*

String

*equals(Object)* boolean *canEqual(Object)* boolean *hashCode()* int

*toString()* String

*valid*

boolean

*dateTime* LocalDateTime

*errorReason* ErrorReason

*id*

long

*sale id place*

Sale long int

*sellingDate theater spectacle id*

LocalDate Theater Spectacle long

*spectacles* Set<Spectacle>

*numberPlaces sales*

*id*

int Set<Sale> long

name() String

staticConstructor() String

onConstructor() ~~AnyAnnotation~~[] access() AccessLevel

String

staticName()

onConstructor() A nyAnnotation[] access() AccessLevel

force() boolean

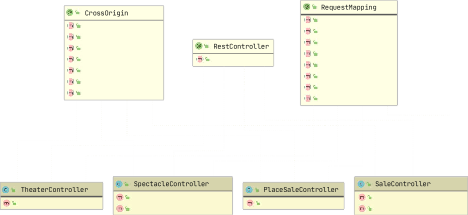
String

staticName()

uniqueConstraints() UniqueConstraint[] indexes() Index[]

String String String

name() catalog() schema()



value() origins()

String[] String[]

allowedHeaders() String[] exposedHeaders() String[] methods() RequestMethod[] allowCredentials() String maxAge() long

name() value() path()

String String[] String[]

value()

String

method() RequestMethod[] params() String[]

headers() String[]

consumes() String[]

produces() String[]

getNumberOfPlaces(long) int

test()

String

getTakenPlaces(long) List<Integer>

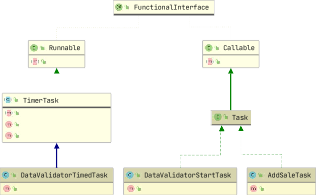
test()

String

test()

String

add(SaleDto) ResponseEntity<?>



run() void

call() V

run() cancel()

void boolean

scheduledExecutionTime() long

run()

void

call()

Object

call() Long