FunAdventure

Link alla repository: https://gitlab.com/daviderendina/2020_assignment3_meet_your_friends

Il progetto è stato svolto in autonomia dallo studente Davide Rendina matricola 830730.

Descrizione dell'applicazione

FunAdventure è un applicativo che permette la gestione degli ingressi in un parco avventura. Ogni cliente per accedere al parco deve registrarsi e, per i clienti minorenni, è necessario specificare quale degli altri clienti registrati nella piattaforma sia il suo accompagnatore (che deve obbligatoriamente essere maggiorenne). Uno o più clienti effettuano un ingresso nel parco, viene fornita loro un attrezzatura (tipicamente il casco) ed effettuano i percorsi desiderati all'interno del parco stesso. Al termine della permanenza di tutti i clienti associati a un particolare ingresso, è possibile effettuare il pagamento dei percorsi effettuati mediante tre tipologie: carta, contanti e voucher.

Configurazione del progetto

Per implementare questo progetto è stato utilizzato il framework Hibernate (al fine di garantire ORM) in combinazione con un istanza di MySQL. In particolare, Hibernate è stato configurato come segue:

- È stata generata una sola PersistentUnit, per tutte le entità persistenti.
- È stato selezionato un valore hibernate.hbm2ddl.auto = update, per far si che le modifiche vengano effettuate sempre sullo stesso db e che lo stesso non venga ad esempio ricreato ad ogni utilizzo del sistema (come accadrebbe ad esempio con un valore create).
- Se non viene trovato nessun database con il nome specificato, esso viene automaticamente creato

Il sistema comunica per il salvataggio delle entità persistenti con un'istanza di MySQL, che presenta la configurazione seguente:

- Rimane in ascolto sulla porta localhost:3306 (default di MySQL)
- Ha un utente con username "funadventure" e password "QYqJcfgMh3#v"

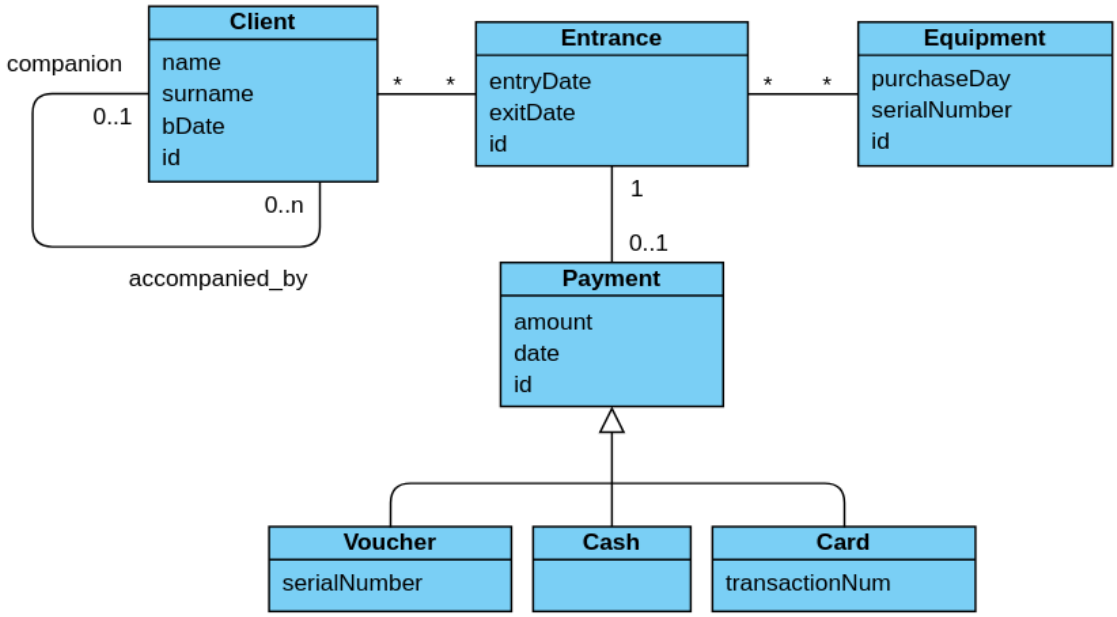
Run del progetto

Per testare il progetto, sono stati implementati appositi test unitari attraverso JUnit. Per eseguirli, è necessario eseguire i seguenti comandi dal terminale in una directory qualsiasi:

- git clone https://gitlab.com/daviderendina/2020_assignment3_meet_your_friends.git
- cd 2020_assignment3_meet_your_friends
- mvn test

Nota bene: per eseguire il progetto, è necessario che l'istanza di MySQL sia in ascolto su localhost:3306. Per lanciare il servizio, è necessario installare MySQL, configurare l'utente con username funadventure e, se necessario, utilizzare i comandi specificati al seguente link per eseguire il server (solitamente, il servizio viene lanciato in automatico).

Diagramma di dominio



Descrizione delle classi

Tutte le entità presentate di seguito (senza considerare le relazioni) sono annotate con @Entity e rappresentano quindi le entità persistenti del sistema. Ogni classe ha un attributo di tipo integer chiamato id, che rappresenta il UID dato dal sistema all'oggetto, che ha funzione di chiave primaria nel database.

Client

Rappresenta un cliente registrato nel sistema del parco avventura, descritto dai suoi dati anagrafici (nome, cognome, data di nascita). Un *Client* può essere in relazione con altri n *Client* (*accompanied_by*): questa relazione indica che il *Client* dalla parte n della relazione è il maggiorenne accompagnatore (*companion*) dell'altro.

Access

Questa entità rappresenta un singolo ingresso che viene effettuato al parco. Un *Access* viene definito solamente da due timestamp relativi alla creazione dell'ingresso (entrata dei clienti nel parco) e alla chiusura e pagamento dello stesso (uscita di tutti i clienti dal parco).
Un *Access* è in relazione con un *Payment* (con cardinalità 1 - 0..1) che rappresenta il pagamento effettuato per lo specifico ingresso; il pagamento è definito con relazione 0 poiché avviene in un momento successivo alla creazione dell'ingresso. Questa entità è in relazione *park_access* e *equipment_used* con rispettivamente i clienti e l'equipaggiamento coinvolti nell'accesso.

Relazione: park_access

Mette in relazione un accesso al parco con i *Client* che l'hanno effettuato. In particolare la relazione è di cardinalità n:n poiché ad un unico accesso al parco possono essere collegati più clienti (ad esempio per i gruppi) mentre un unico cliente può essere presente in diversi ingressi (ad esempio, in giornate diverse).
Per implementarla, è stato deciso di inserire all'interno di *Access* una collezione di *Client*, e rendere così la relazione navigabile in un solo verso.

Relazione: equipment_used

Relazione con cardinalità n:m che mette in relazione un *Access* con tutti gli *Equipment* che sono stati utilizzati in quell'accesso. La cardinalità è n:m poiché in un *Access* potrebbero esserci diversi *Equipment* utilizzati (ad esempio, da clienti diversi) e allo stesso tempo un *Equipment* può essere in relazione con diversi *Access* (in giorni diversi).

Per implementarla, è stato deciso di inserire all'interno di *Access* una collezione di *Equipment*, e rendere così la relazione navigabile in un solo verso.

Equipment

Rappresenta un singolo equipaggiamento (es. casco, imbrago, ..) in possesso del parco avventura. È descritto dalla sua data di acquisto e dal numero seriale.

Payment

Descrive il pagamento effettuato dai clienti nei confronti del parco al termine della loro visita. È descritto da un campo float *amount*, che rappresenta il conto saldato, e dalla data e ora del pagamento. *Payment* è una generalizzazione di tre diverse entità, che rappresentano i diversi tipi di pagamento possibili: *Voucher*, *CreditCard* e *Cash*.

VoucherPayment

Rappresenta il pagamento effettuato tramite voucher dal cliente. Un pagamento di questo tipo è descritto anche dal numero seriale del voucher utilizzato dal cliente.

CardPayment

Rappresenta il pagamento effettuato tramite carta/bancomat, e aggiunge un nuovo campo che contiene il numero di transazione del pagamento.

CashPayment

Rappresenta il pagamento effettuato tramite contanti.

Architettura e package

L'applicazione è stata divisa in 4 package principali:

1. Domain
2. Service
3. Repository
4. Utils

Domain

Il package *domain* contiene tutte le classi (descritte in precedenza) che rappresentano le entità del dominio (i POJO).
Ogni entità è stata mappata attraverso le annotazioni JPA con una tabella del database.

Gestione di un Payment

Per la gestione del campo *Payment* all'interno dell'*Access*, è stato scelto di annotare il primo con un valore di CascadeType=ALL, per far sì che ogni operazione CRUD effettuata sull'*Access* sia applicata anche all'entità *Payment*. Questa scelta è stata effettuata poiché il ciclo di vita di un *Payment* è strettamente legato all'*Access* che lo contiene, e questa annotazione permette quindi di semplificare la sua gestione (poiché si occupa il framework di definire anche le operazioni necessarie per il *Payment*, quando si opera su un *Access*).

Service

Questo package rappresenta lo strato Service dell'applicazione, contenente tutta la logica di businness del sistema. Per ogni entità (*Client*, *Access*, *Equipment* e *Payment*) è presente una classe Service, che implementa la logica e comunica con lo strato di *Repository* per garantire la persistenza delle entità. In particolare, ogni Service contiene un oggetto Repository corrispondente per delegare le varie operazioni.

Ogni service contiene i metodi per la creazione, modifica, cancellazione e ricerca di entità (ricerca per id, ricerca di tutte le entità di quel tipo e eventualmente altre search più sofisticate).

Al fine di facilitare l'accesso ai servizi forniti da questo strato, è presente una *ServiceFacade* che ha il compito di esporre la API per utilizzare lo strato.

Questo strato del sistema contiene anche la classe *PaymentFactory*, una Factory che si occupa di creare gli oggetti di tipo *Payment*.

PaymentService

Per *PaymentService* è stato definito un comportamento diverso dalle altre entità. Anche se *PaymentRepository* implementa tutte le operazioni CRUD, il Service implementa solamente le funzioni di ricerca (find), poiché per le altre operazioni (Create, Update, Delete) il *Payment* è gestito insieme all'*Access* che lo contiene come visto in precedenza.

Repository

Questo package rappresenta lo strato di Repository dell'applicazione, che ha lo scopo di interfacciarsi con il database per effettuare le operazioni CRUD sulle entità. Per ogni entità (*Client*, *Access*, *Equipment* e *Payment*) è presente una diversa repository e ognuna di esse implementa un'interfaccia *Repository*, che ha il compito di uniformare le API di accesso alla persistenza per tutte le istanze.

Per comunicare con la base di dati, ogni repository ha al suo interno un istanza di EntityManager, creata attraverso un *EntityManagerFactory*. Siccome la creazione e il mantenimento di un'istanza di tipo *EntityManagerFactory* è molto dispendiosa, è stato scelto di implementare una classe *RepositoryFactory*. Questa classe ha la responsabilità di creare il repository richiesto passandogli, attraverso il costruttore, un oggetto di tipo *EntityManagerFactory*, per la creazione dell'EntityManager. In questo modo viene creata e mantenuta aperta un'unica istanza di questo tipo per tutte le Repository.

Utils

Contiene solamente la classe *IdGenerator*, che ha la responsabilità di generare gli UID per tutti gli oggetti del sistema.