Avogadro Roberto 807387 – Belotti Federico 808708 – Carta Costantino 808417

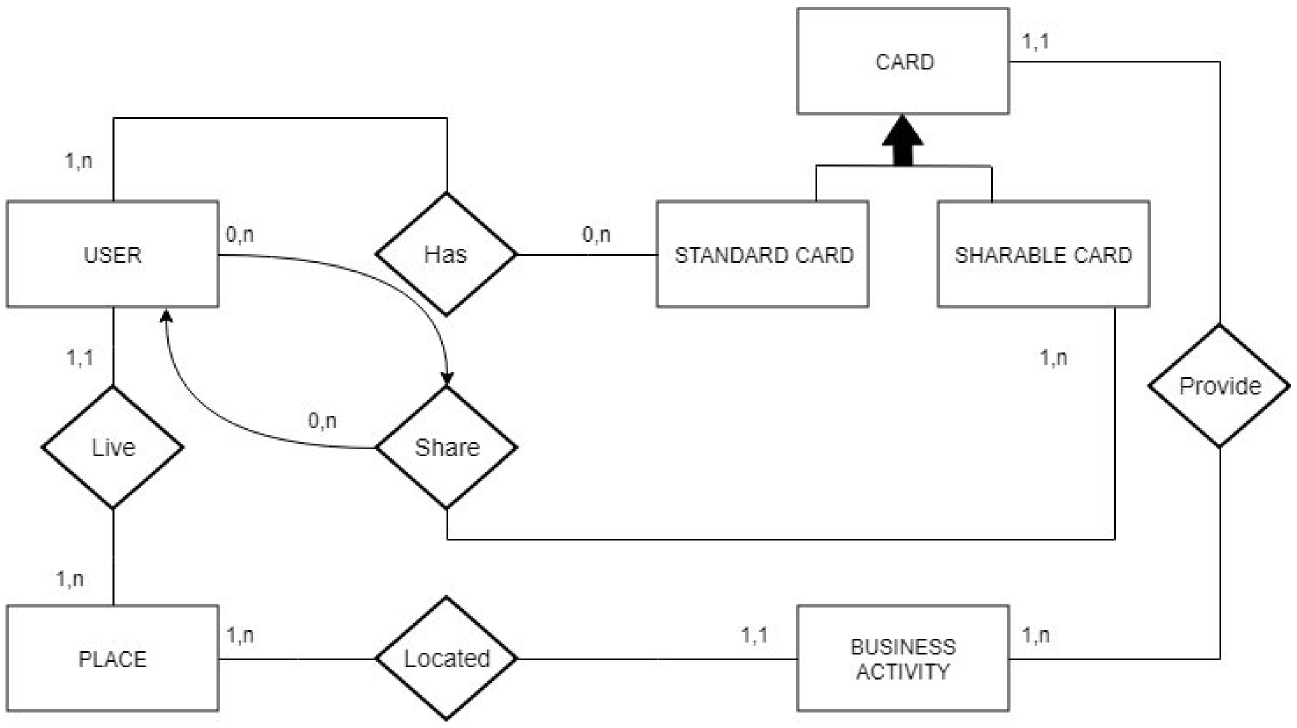
Assignment 3 – first submission

Link Repository:
https://gitlab.com/belerico/prosviso_assignment_3
Link README:
https://gitlab.com/belerico/prosviso_assignment_3/blob/master/README.md

Application overview

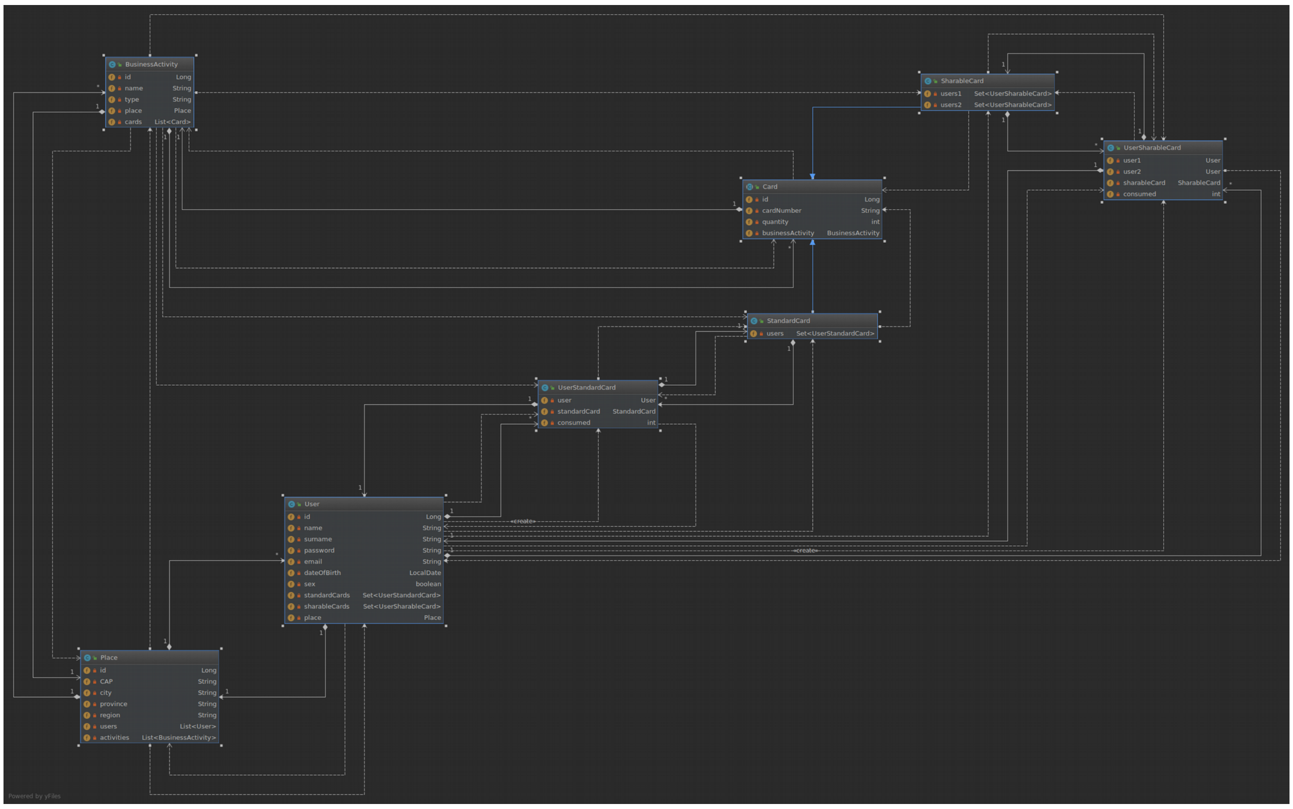
The purpose of this application is to allow users to use virtual cards in order to remember that they have promotional offers pending (e.g if you eat 10 pizza you'll get one for free). Most of the time we forget that we have this kind of cards in our wallet, so we want to help user in that way.
In other hand we'll help also business activity as they'll get more visibility standing on the application.
Instead of the paper card we will provide an innovative way to exploit this promotions.

E-R Schema



The **E-R** schema shows relationships and roles between entities, so:
- A user can have as many as standard card he wants, so a standard card can be used by many users.
- A user can share a sharable card with another user
- Cards are made available by a business activity, which decides how many cards to provide

Class diagram UML



This class diagram has been created with IntelliJ IDEA help

Class Description

Our base package is com.assignment3 in which are contained the packages:
-jpa
-struts2
-utils

Jpa package contains three packages called:
-DAO
-model
-service

The **DAO** package contains the classes who manage CRUD operations for the various entity using the DAO design pattern.

- A DAO interface has been defined which specifies the operations that each DAO(one for each entity) has to implement.
- An abstract class *AbstractDAO* has been defined which implements the DAO interface and defines the implementation of every interface’s methods. Note that even if this class has been defined as Abstract in reality doesn’t contains any abstract method. The motivation of this chosen is due to the fact that for reason of application logic we don’t want to allow the creation of an AbstractDAO type object;
- DAO classes defined for each entity (*BusinessActivityDAO, CardDAO,*

UserDAO, PlaceDAO) inherit from *AbstractDAO* class;
- The *EntityManagerSingleton* class adopts the Singleton design pattern and it has been defined to have a singleton of the classes EntityManagerFactory and EntityManager instances;

The **model** package contains *JPA* classes for each entity of the developed system. Each class is a JPA entity (specified with **@Entity** annotation) and it’s mapped in the database.
It was therefore necessary to create UserStandardCard and UserSharableCard classes who don’t represent any entity within our system, but they map the **@ManyToMany** bidirectional relationship in which there are the relationship attributes. We had to create these two classes as for both of them we need:
1. The quantity extra column
2. Performance. As described in the Hibernate documentation mapping a **@ManyToMany** relationship as two **@OneToMany-@ManyToOne** it would improve a DELETE query from the link table

The **service** package contain ‘service’ classes which abstract the use of DAO and moreover the provide high level operations for the management of the entities.

- A Service interface has been defined which specifies the operations which each specific Service (one for each entity) has to implement;
- An abstract class *AbstractService* has been defined which implements the Service interface and define the implementation of every interface’s method apart *‘delete(T entity)’* method which will be implemented by every specific service;
- The Service classes defined for each entity (*BusinessActivityService, CardService, UserService, PlaceService*) inherit from *AbstractService* class;
- The *ServiceFactory* class adopt the Singleton design pattern for each service and it has been defined to have one singleton of the classes *BusinessActivityService, CardService, UserService, PlaceService* istances

The **struts2** package contains four packages:
- **action**
- **converter**
- **listener**
- **validator**

The **action** package in turn contains four packages(one for each entity):
- **activity**
- **card**
- **pace**
- **user**

Every package within **action** package contains the controllers of the architectural pattern **MVC**(model-view-controller) used in the web application logic. Furthermore in the **action** package there is *RedirectAction* class which works as controller to redirect to a page which doesn’t have any associated action (e.g Index.jsp).

The converter package contains the LocalDateConverter class which is used to convert the date typed by the user into a LocalDate type field following the ISO-8601 calendar system.

The **validator** package contains the *LocalDateValidator* which is used to validate a LocalDate type field, if correctly converted by the LocalDateConverter.

N.B : we intentionally wanted to separate the field validation logic from the controller/activity logic, so we’ve created in src/main/resources/com/assignment3/struts2/action a directory, for each package in com.assignment3.struts2.action, which contains the validation.xml file as specified by the struts2 documentation. So, for example: we wanted to validate the user creation (every field must not be empty, the email must be unique, the date must follow yyyy-mm-dd format and so on), so in src/main/resources/com/assignment3/struts2/action/user there is a file called UserAction-createUser-validation.xml (ActionClassName-methodToValidate-validation.xml).

N.B : converter and validator package were made as struts2 works with Date type field which was unmatchable in our project as we needed LocalDate type field (plus struts2 validation works when it wants).

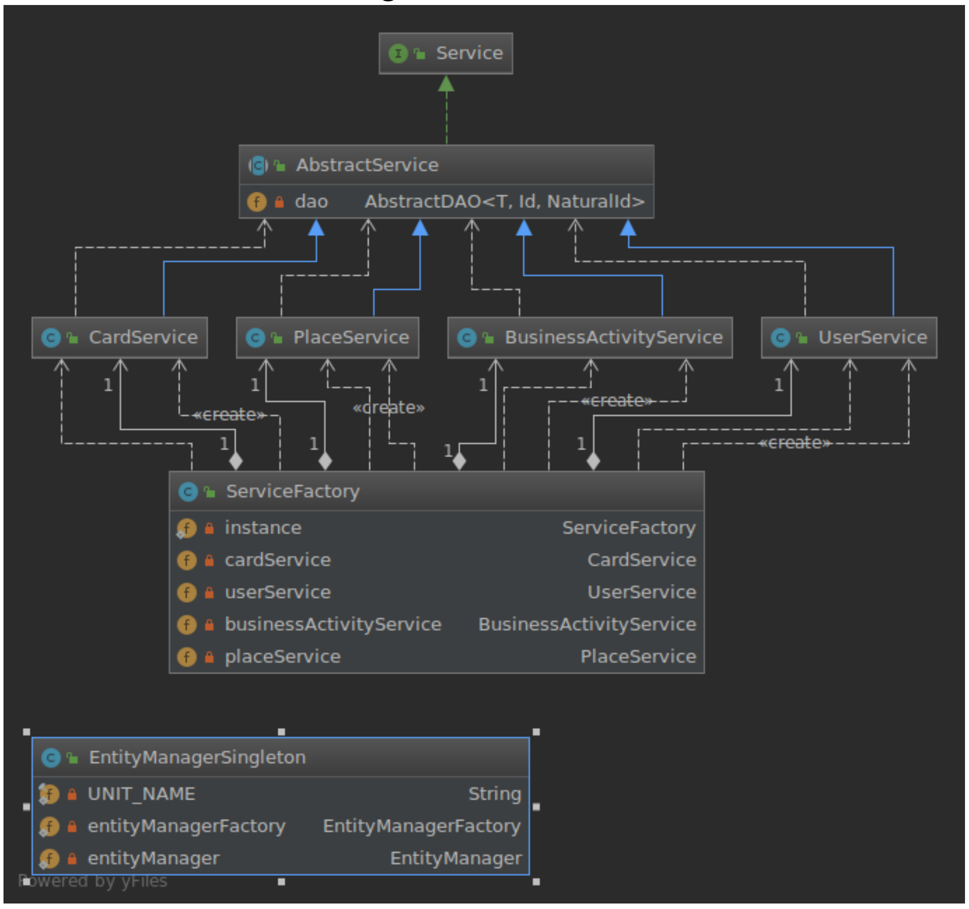
The **listener** package contains the *WebappServletContextListener* class which purpose is to drop database as server starts running and furthermore it populates the database with fake entities.

The **utils** package contains a **faker** package and furthermore contains an *Helper* class which allow to perform the population and the drop of the database. *Helper* class is used for example as the webapp starts running.

The **faker** package contains a class for each entity of the developed system that are used to create fake entities.

The **webapp** package refers to the web-side of the developed system; it contains the views of the MVC architectural pattern. In particular there are views for each entity of the developed system.

Service model diagram


Service
AbstractService
dao AbstractDAO<T, Id, NaturalId>
CardService
PlaceService
BusinessActivityService
UserService
1
«create»
ServiceFactory
instance ServiceFactory
cardService CardService
userService UserService
businessActivityService BusinessActivityService
placeService PlaceService
EntityManagerSingleton
UNIT_NAME String
entityManagerFactory EntityManagerFactory
entityManager EntityManager
Powered by yFiles