Lab Assignment #3

Fall Semester 2019

March 27th

Repository available at `https://classroom.github.com/g/UsdgOweo`

1 Introduction

This third lab assignment will be graded and you will be working in pairs, so team up with a peer to start working. We expect from you to only be working with your peer and avoid collaborating with others during the lab. Your work must conform to the *Code of Academic Ethics* included at the end of this document.

You will be experimenting with the ActiveRecord Object-Relational Mapping (ORM) framework to implement the model layer of a web application. By the end of the assignment, you are required to submit a working Ruby on Rails application containing the model layer, that is, all models and associations, fully developed.

2 Description

You will create the data model for a web application that implements a ticket sales system. Systems of such kind usually present the user an event catalog, and the user reviews event information and purchases tickets. The data model for the system must comprise the following features:

- Persist user information as name, last name, email, phone, password and address.
- Persist event venue information as name, address and capacity.
- Persist events information as name, description, start date, and in which an event takes place.
- Persist ticket type information including event, price and category ("Pacifico", "Andes", "Galeria" and so on).
- Persist ticket orders, including the total amount payed.

You will test the models in your application by using the console provided by the RoR framework (`rails c`).

3 Goals

You need to accomplish the following goals by the end of the assignment:

1. Define a domain model for the application that responds to the requirements listed above. Consider that all the information about users, places, events and tickets must be persisted. The domain model must comprise a textual description of the entities involved, and their relationships. Along with the textual description, you may use Axure to sketch your domain model, including cardinalities in the associations.

2. Create the models in your application according to the domain model

   (a) Create all the models needed by the domain [1.5 pt.]
   It is possible to create all the needed models and their associations by running the following commands in the terminal:

   ```
   rails g model User name:string lastname:string email:string password:string address:string
   rails g model EventVenue name:string address:string capacity:integer
   rails g model Event name:string description:string start_date:date event_venue:references
   rails g model TicketZone zone:string
   rails g model TicketType event:references price:integer ticket_zone:references
   rails g model Order user:references
   rails g model Ticket ticket_type:references order:references
   ```

   (b) Add the associations corresponding on the ruby models [1.5 pt.]

   ```
   class User < ApplicationRecord
     has_many :orders
   end
   class Order < ApplicationRecord
     belongs_to :user
     has_many :tickets
   end
   class TicketType < ApplicationRecord
     belongs_to :event
     belongs_to :ticket_zone
   end
   class TicketZone < ApplicationRecord
     has_many :ticket_type
   end
   class Ticket < ApplicationRecord
     belongs_to :ticket_type
     belongs_to :order
   end
   class EventVenue < ApplicationRecord
   end
   class Event < ApplicationRecord
     belongs_to :event_venue
     has_many :ticket_types
   end
   ```

3. Populate the local database [1 pt.]

(a) Edit the `db/seeds.rb` file to initialize your data base with sample data `rails db:seed`

(b) Create five different users.

(c) Create three different events.

(d) Each event must have at least three ticket categories.

(e) Each user must have bought one ticket for each event.

```
users = [User.new(name:'Claudio', lastname:'Alvarez', \
             password:'123456', email:'calvarez1@miuandes.cl', \
             address:'San Carlos de Apoquindo'),
         User.new(name:'Juan', lastname:'Rataplan', \
             password:'123456', email:'jrataplan@miuandes.cl', \
             address:'San Carlos de Apoquindo'),
         User.new(name:'Raul', lastname:'Rabufetti', \
             password:'123456', email:'rrabufetti@miuandes.cl', \
             address:'San Carlos de Apoquindo'),
         User.new(name:'Raul', lastname:'Ganfolfi', \
             password:'123456', email:'rgandolfi@miuandes.cl', \
             address:'San Carlos de Apoquindo'),
         User.new(name:'Licenciado', lastname:'Varela', \
             password:'123456', email:'lvarela@miuandes.cl', \
             address:'San Carlos de Apoquindo')]

for u in users do
  u.save!
end

event_venues = [EventVenue.new(name: 'Estadio Nacional', address:'Maraton', capacity: 60000),\
                EventVenue.new(name: 'Estadio Monumental', address:'Exequiel', capacity: 25000),\
                EventVenue.new(name: 'Movistar Arena', address:'Matta', capacity: 15000)]

for ev in event_venues do
  ev.save!
end

events = [Event.new(name:'Festival de la Cancion', description: 'Puro reggaeton',\
          start_date: '2019-03-01', event_venue: event_venues[0]),
          Event.new(name:'Twisted Sister', description: 'Puro rock',\
          start_date: '2019-04-01', event_venue: event_venues[1]),
          Event.new(name:'Bad Bunny', description: 'Puro Trap',\
          start_date: '2019-05-01', event_venue: event_venues[2])]

for e in events do
  e.save!
end

ticket_zones = [TicketZone.new(zone: 'Cancha'),\
                TicketZone.new(zone: 'Pacifico'),\
                TicketZone.new(zone: 'Pacifico VIP')]

for tz in ticket_zones do
  tz.save!
end

prices = [20000, 30000, 40000]
```

```
ticket_types = []

for e in events do
  i = 0
  for t in ticket_zones do
      ticket_types << TicketType.new(price: prices[i], ticket_zone: t, event: e)
      i += 1
  end
end

for tt in ticket_types do
  tt.save!
end

for u in users do
  for e in events do
    o = Order.new(user: u)
    o.save!
    tts = TicketType.where(event:e)
    tt = tts.sample # get any ticket type available
    t = Ticket.new(order:o, ticket_type:tt)
    t.save!
  end
end
```

4. Add some custom behavior to your models

   (a) User [1 pt.]

      i. Instance method `most_expensive_ticket_bought`, which returns the price of the most expensive ticket bought by the user.

      ii. Instance method `most_expensive_ticket_bought_between`, which receives two dates (start and end dates) as arguments and returns the most expensive ticket bought by the user between the specified dates.

      iii. Instance method `last_event`, which returns the name of the last event for which the user bought a ticket.

```
class User < ApplicationRecord
  has_many :orders

  def most_expensive_ticket_bought
    # Look for self's orders, then the corresponding tickets, the the corresponding ticket types
    # and finally, look up the maximum price
    TicketType.where(id: Ticket.where(order: self.orders).select("ticket_type_id")).maximum(:price)
  end

  def most_expensive_ticket_bought_between(date_start, date_end)
    # The same as the above, but pre-filter the purchases according to the dates given
    ods = self.orders.where("created_at >= ? and created_at <= ?", date_start, date_end)
    TicketType.where(id: Ticket.where(order: ods).select("ticket_type_id")).maximum(:price)
  end

  def last_event()
```

```
    # Get all the events for which the user has purchased tickets, and select the one
    # with the latest start date
    Event.where(id: TicketType.where(id: Ticket.where(order: self.orders).\
      select("ticket_type_id")).select("event_id")).order(start_date: :desc).first
  end
end
```

(b) Venue [1 pt.]

i. Instance method `last_attendance`, which returns the number of attendants (i.e., people who bought a ticket) in the last event that was hosted at the venue.

```
class EventVenue < ApplicationRecord
  def last_attendance
    # Latest event hosted at this venue
    last_event = Event.where(event_venue:self).order(start_date: :desc).first

    # Get all the corresponding tickets
    Ticket.where(ticket_type: TicketType.where(event: last_event)).count
  end
end
```

(c) Event [1 pt.] (BONUS)

i. Class method `most_tickets_sold`, which returns the event for which the most tickets have been sold.

ii. Class method `highest_revenue`, which returns the event with highest revenue.

```
class Event < ApplicationRecord
  belongs_to :event_venue
  has_many :ticket_types

  def self.most_tickets_sold
    # The easiest and most efficient in this case is to simply issue a single SQL query
    sql = "select event_id, count(*) as c from tickets t, ticket_types tt \
    where t.ticket_type_id = tt.id group by tt.event_id order by c desc limit 1"
    result = ActiveRecord::Base.connection.execute(sql)
    Event.find(result[0]['event_id']) unless result.nil?
  end

  def self.highest_revenue
    # Same approach as the above
    sql = "select sum(tt.price) as s, tt.event_id from tickets t, ticket_types tt \
    where tt.id = t.ticket_type_id group by tt.event_id order by s desc limit 1"
    result = ActiveRecord::Base.connection.execute(sql)
    Event.find(result[0]['event_id']) unless result.nil?
  end
end
```

4 Grading

Your grade will be computed based on your accomplishments, considering the scores listed above for every milestone. Consider six points plus the base one.

5 Testing your application

For the purpose of testing your app, you can use the RoR console to interact with your models and create instances of them, and even executing the custom behavior (methods) of your models, to see if the return values correspond to what you expect in each case.

For example, running the following command, after inserting some values into your tables, will show you all the tables existing in your DB:

```
$ rails db
sqlite> .tables #output the tables names
```

Now, for playing with your models you should execute the rails console:

```
$ rails c
>> User.create(arguments) #this will create a new User record
>> User.all #will return all the user records in the db
```

You can see more details of the queries you can execute on the references listed below, in particular, "Active Record Queries".

6 Useful links

The following links to Rails Guides will provide you useful information for completing your assignment:

- Command line: `http://guides.rubyonrails.org/command_line.html`
- Active Record Basics: `http://guides.rubyonrails.org/active_record_basics.html`
- Active Record Model: `http://api.rubyonrails.org/classes/ActiveModel/Model.html`
- Basic Models Associations: `http://guides.rubyonrails.org/association_basics.html`
- Active Record Association Methods: `http://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html`
- Active Record Migrations: `http://edgeguides.rubyonrails.org/active_record_migrations.html`
- Active Record Queries: `http://guides.rubyonrails.org/active_record_querying.html`

7 Repository Setup

First, you need to sign up at the classroom assignment on Github, following the same steps as last week. Go to the Github classroom assignment at `https://classroom.github.com/g/UsdgOweo` (copy and paste this URL in your web browser) and accept the invitation to the assignment.

After your personal repository has been created, do a git clone specifying the url provided by Github, to your Desktop location in you computer. You will find the clone url in the button "Clone or Download" located at your repository profile.

You need to add a README.md file to you repository as your first commit. That file must contain the name of the two persons working in the project. Please provide your full name. The commit message must be "README added".

After this, you will have to create a new rails project inside the repository you just cloned:

```
cd <repository path>
rails new .
```

8 Code of Academy Ethics

During the course and for every task, test and homework given to you, all the ethical criteria established by the Faculty of Engineering and Applied Sciences at the Universidad de los Andes, Chile will apply:

> "Any detection of copying, plagiarism, or dishonest behavior, independent from the fact itself, will be reviewed by the Faculty Council. The minimum sanction will be a 1.0 as the final grade of the course, with the possibility of escalating to the expelling of the student from the University.
>
> Any Student surprised in an activity that calls the atention of the professors and assistants, will be punished as if he/she had commited or executed any of the dishonest behaviors. In other words, it is exclusively the responsabilty of the students to behave correctly and meaningfully."