CS 169 Fall 2019 - Week 9 - Advanced Rails

https://github.com/skleung/Community - Kevin's app he demo'd in section.
Filters - app/controllers/application_controller.rb - note the skip_before_filter in the welcome_controller.rb
Validations - app/models/diner.rb has `validates_presence_of :name` although it might not be such a great example b/c of the before_validation function running (this was so people could sign up through venmo without specifying a name)
app/models/group.rb is probably an easier validation example since a lot of business logic is going on in diner.
Associations - basically all models - app/models/*

Validation and Filters
Validation and filters are two examples inside of Rails of aspect oriented programming. Validations are applied to models, and are use to check certain conditions before allowing a model to save data to the database. Filters on the other hand are used to check certain conditions before allowing a controller action to run.

Writing a Validation
Say we have a `User` model as follows:

```
class User < ActiveRecord::Base
      validates :username, :presence => true
      validate :username_format
end
```

What happens if we have `@user` with no username and we call `@user.valid?`. What will `@user.save` do? What will `@user.save!` do?
`@user.valid? returns false, @user.save returns false and won't save to the database, @user.save! will thrown an exception and won't save to the database.`
Implement `username_format`. For our purposes, an username starts with a letter and is at most 10 characters long. Remember, custom validations add a message to the errors collection.

```
def username_format
      if username.length < 10 or not username =~ /^[a-z]/i
            errors.add(:username, "is not formatted correctly")
      end
end
```

Using Filters
Say we wanted to check if `@user` was an admin for all the methods on the `AdminController`. Write a `before_filter` that checks if the admin field on `@user` is true, and if not, redirects to the `'/admin_login'` page with a message.

```
class AdminController < ApplicationController
      before_filter :check_admin
      def check_admin
            if not @user.admin
                  flash[:notice] = "You must be an admin"
                  redirect_to '/admin_login'
            end
      end
end
```

Associations

Associations are a powerful tool inside of rails that allow us to define relationships between models. Rails hides away a lot of the complications between making joins and other database operations, which makes life a lot easier.

Setting up Associations

For each group of models, describe what association you would add to each model and what migrations you would need to run to make the methods work.

a. `@farmer.cows`
   `Farmer has_many cows, need foreign key on cow`

b. `@pokemon.trainer` and `@trainer.pokemons`
   `Pokemon belongs_to trainer, Trainer has_many pokemon, key on pokemon`

c. `@student.majors, @major.students, @student.degrees, @major.degrees, @degree.major, @degree.student`
   `Students has_many majors through degree, has_many degrees`
   `Major has_many student through degree, has_many degrees`
   `Degree belongs_to major, student, has foreign key`

https://www.dropbox.com/s/9cjhy6s21kzvf6g/2015-02-18%2023.25.09.jpg?dl=0

Life Without Associations

We want to model a one to many relationship between `User` and `Picture`; i.e. a user can own many pictures, and a picture has one owner. To do this, we added a foreign key for users onto pictures (so pictures have a field `user_id`).

How would we implement the following actions WITHOUT having `belongs_to` and `has_many` on our models.

a. Create a new Picture that belongs to `@user`.
   `Picture.create(user_id: @user.id)`

b. Delete `@user` and all of of the pictures associated with that user.

```
@pictures = Picture.where(user_id: @user.id)
@pictures.each do |picture|
     picture.destroy
end
@user.destroy
```

Now say we added `belongs_to` and `has_many` to their respective models. How would implement the two actions above?

```
@user.pictures.create
@user.pictures.destroy_all
@user.destroy (better is to add dependent: destroy)
```