Lab 9.1: Embedding and Referencing in Models with Mongoid
IN705 Databases Three

Introduction

Last week we set up MongoDB and modified our `User` model to persist to it. We were able to perform basic CRUD operations on our user model without any modification to our `UserController`. Other operations, however, like creating splatts and setting up follower/followed relationships between users will require a bit more machinery that we will explore in this lab.

1 Creating a new Splatt model class

Splatts belong to Users. Our original model class used a *foreign key* to indicate which user owned a particular splatt. We could do something very similar with our MongoDB version, but instead we'll do something more direct. Since Splatts belong to Users, why not iunclude them directly in the User document records? This wasn't an option with the relational model since a relational database table can't store a variable length list of records. In a document store like MongoDB this is easy.

Our new Splatt class looks like this:

File: app/models/splatt.rb

```
class Splatt
  include Mongoid::Document
  include Mongoid::Timestamps::Created
  field :body, type: String
  embedded_in :user
end
```

Let's go through it line-by-line:

`include Mongoid::Document` This allows us to save model data to and retrieve it from a MongoDB document store.

`include Mongoid::Timestamps::Created` Unlike ActiveRecord, Mongoid does not automatically include timestamps in its models. We use the `created_at` timestamp with our Splatts, so we include support for it here.

`field :body, type: String` The main feature of a Splatt is that it contains string content that we declare here.

`embedded_in :user` This tells Mongoid that Splatt instances will actually be stored as a list contained in the user collection. We must also indicate the other end of the relationship in the `User` class.

To indicate the other end of the embedding relationship, add the following line to the User class:

File: app/models/user.rb

```
embeds_many :splatts
```

2 Changes to the SplattsController

We have changed our Splatts model a bit since it no longer includes a foreign key. Instead we create a new Splatt instance and add it to a particular user's list. The modified `create` method in the controller looks like this:

File: app/controllers/splatts_controller.rb

```
 def create
     @user = User.find(params[:user_id])
     splatt = Splatt.new({:body => params[:body]})
     if @user.splatts.push(splatt)
           render json: splatt, status: :created, location: @user
     else
           render json: @user.errors, status: :unprocessable_entity
     end
end
```

Since Splatt instances are embedded in User documents now, we have to load the user from the database, create the new splatt with the specified body, then push the new splatt onto the user's embedded list.

3 Implementing following with references

In our relational models we implemented the follower/followed relationships with a self-join through an intermediate joining table that held the appropriate user ids. Documents in MongoDB all have unique ids that we can use in a similar way, but we don't need the intermediate table. Instead, each user document will contain lists of followers and users followed. Mongoid uses the same `has_and_belongs_to_many` association we saw in ActiveRecord.

To implement following in your user model, add the following two lines:

File app/models/user.rb

```
 has_and_belongs_to_many :follows, class_name: "User"
 has_and_belongs_to_many :followers, class_name: "User"
```

The contoller methods used for adding and deleteing follows must be modified as well, since we need to explicitly control the associated followers and follows lists. For example, the `add_follows` method new looks like this:

File: app/controllers/users_ controller.rb

```
def add_follows
    @user = User.find(params[:id])
    @follows = User.find(params[:follows_id])

    if @user.follows << @follows and @follows.followers << @user
          head :no_content
 render json: @user.follows
    else
          render json: @user.errors, status: :unprocessable_entity
    end
end
```

First, we create the user who will do the following and the user who will be followed. We push the followed user onto the first user's `follows` list (really we just push that user's id). The we push the follower onto the follwoed user's `followers` list.

The `delete_follows` method is similar and is left as an exercise.

4 Test and insepct your saved data

With your changes mades you should be able to create and modify users, enter splatts, and follow/unfollow users. Using `curl`, create some users in your database, enter some splatts for them, and do some following. Then, connect to your MongoDB database and inspect the resulting user documents. Notice how our MongoDB database only uses one collection `users`, instead of the three tables that our remational model required.