Lab 12.1: Data Modeling with Riak
IN705 Databases Three

Introduction

Last week we discussed how we could organise our data models for a key–value store like Riak. We also saw that the modeling libraries for Riak do not have some of the nice features of libraries like ActiveRecord and Mongoid. We will have to take a more hands on approach to working with the database when using Riak.

1 Setup

If you have not already done so, create a new working directory called '`riak`' in your home directory on your EC2 server. Inside that directory, clone your github repository (e.g.: `git clone git@github.com:username/db3.git`). Then, create a new branch with the command `git branch -b riak`. This creates a new branch called '`riak`' ans switches to it.

Next, open your `Gemfile` and add the following lines to it.

file: Gemfile

```
gem 'riak-client'
gem 'hashie'
```

Note that we will leave the line including the `sqlite3` gem in our Gemfiles for now. This will allow us to replace and test the database code in small increments rather than taking an all or nothing approach. For this reason you also need to invoke `rake db:migrate` even though we don't intend to use the Sqlite database it creates.

2 Creating the user model

Our intent is to create a `User` object model that only concerns itself with presenting user data to the application. Database persistence will be handled by the `UserRepository` class below. This means that our `User` class can be quite simple. We will use the `HAshie::Dash` library to help organise our model and to handle JSON serialisation and deserialisation.

Our `User` model needs to store

- email
- name
- password
- blurb

- follows
- followers

We decided last week that we would use email addresses as our keys, and that follows and followers are both lists of keys (email addresses).

So, our new `User` class looks like this:

app/models/user.rb

```
class User < Hashie::Dash
  property :email
  property :name
  property :password
  property :blurb
  property :follows
  property :followers
end
```

Now we need a new class to handle moving user data to and from the data store. In the past this was handled by libraries we used, but in this case we need to write that ourselves. We will do this by writing a companion class for our `User` class called `UserRepository`.

To see what method our `UserRepository` will implement, we look at the `UsersController` class to see how we access our data store. We see that our repository needs `find`, `save`, `update`, and `delete` methods. There are some additional methods that we need, but we will start with this list.

Start by creating a skeleton for our `UserRepository` class:

file: app/models/user_ repository.rb

```
class UserRepository
  BUCKET = 'users'

  # sets up our connection to the Riak db
  def initialize(client)
    @client = client
  end

  def all

  end

  def delete(user)

  end

  def find(key)

  end

  def save(user)

  end

  def update(user)

  end
```

```
end
```

We will start by defining `save` and `find` so that we can create a new user and then see it.

Our `save` will take a `User` object as an argument. It should check to see that the user record does not already exist and save the record if it does not. The resulting method looks like this:

```
def save(user)
  users = @client.bucket(BUCKET)
  key = user.email

   unless users.exists?(key)
     riak_obj = users.new(key)
     riak_obj.data = user
     riak_obj.content_type = 'application/json'
     riak_obj.store
   end
end
```

Our `find` method is pretty simple. It gets the data from the database and populates a `User` object with it. This initial version is a little brittle, since it will not recover gracefully if we ask for a nonexisitent user.

```
def find(key)
  riak_obj = @client.bucket(BUCKET)[key]
  user = User.new
  user.email = riak_obj.data['email']
  user.name = riak_obj.data['name']
  user.password = riak_obj.data['password']
  user.blurb = riak_obj.data['blurb']
  user.follows = riak_obj.data['follows']
  user.followers = riak_obj.data['followers']
  user
end
```

Note that there are some issues with the follows/followers properties that we will defer until next time.

Our controller methods need some modification as well. The `create` method will create a `User` object, and then save it with a `UserRepository` object.

```
def create
  @user = User.new
  @user.email = params[:email]
  @user.name = params[:name]
  @user.password = params[:password]
  @user.blurb = params[:blurb]

  db = UserRepository.new(Riak::Client.new)
  if db.save(@user)
     render json: @user, status: :created, location: @user
  else
      render json: "error", status: :unprocessable_entity
  end
end
```

Our show method is straightforward:

```
def show
```

```
  db = UserRepository.new(Riak::Client.new)
  @user = db.find(params[:id])
  render json: @user
end
```

Once you have these methods working correctly, try to implement `all`, `update`, and `delete`. You will probably need to consult the `riak-client` information at https://github.com/basho/riak-ruby-client.