IWDD100

Using ActiveRecord with Sinatra

Necessary gems

```
gem 'activerecord'
gem 'sinatra-activerecord'
gem 'sqlite3'
gem 'rake'
```

Database Setup

```
require ‘sinatra’
require ‘sinatra/activerecord’

set :database, “sqlite3:///<databasename>.sqlite3
```

Rake

- A command line task-runner
- Runs small Ruby tasks written in scripts from the command line, for instance:
  $ `rake db:migrate`
- Database tasks are run by Rake

Rakefile

- Your Rakefile typically contains tasks that you've defined
- For our purposes, we're going to include tasks that the `sinatra/activerecord` library has defined to manage the database
- We'll also include your main app file so the Rakefile is aware of your app and the sinatra/activerecord library require line

Rakefile

```
require 'sinatra/activerecord/rake'
require './app'
```

Migrations

- A migration file is a small Ruby script meant to make changes to your app's database
- Migrations are not typically used to add actual data
- Instead, they're mostly used to set up the structure of your database:
  - to create new tables or rename them
  - to drop a table
  - to add a column to an existing table

Migrations

- To generate a new migration file, use the create_migration Rake task:

```
rake db:create_migration NAME=create_users_table
```

- NAME should be set to something descriptive, but never the name of the table you're creating

Working with Migrations

- A migration is just a Ruby class with boilerplate methods meant to run when the migration is run
- Edit it by going to the db/migrate folder generated when you run the create_migration task
- Add your instructions to the `change` method to have them run when the migration is run

Working with Migrations

```
1  class CreateUsersTable < ActiveRecord::Migration
2    def change
3      create_table :users do |t|
4        t.string :email
5        t.datetime :birthday
6      end
7    end
8  end
9
```

- Inside of the change method, we've called the `create_table` method, giving it `:users` as an argument for our table name
- The `|t|` and block syntax (`do`) indicates that we're using `t` to set further options
- In this case, our further options are different columns by data type, which take a name for the column as an argument

Working with Migrations

- Now that our migration has been written, we can run it from the command line with:

```
rake db:migrate
```

- Don't forget that just changing the migration file itself without running a rake command will do **nothing** to your database

Models

- In order to access the table that you just created, you have to have a class to represent it extended from ActiveRecord::Base

```
#in models.rb
class User < ActiveRecord::Base
end
```

- Extending from another class gives you all of its methods “for free”

- In this case, we get all of the methods to easily access the database

Models

- Make sure you require your newly minted models.rb file in your main app file, right after your line to set the database:

  ```
  require ‘./models’
  ```

- To try out your shiny new `User` model, load up IRB in the terminal, require your main app file, and try typing the name of the class, `User`

```
activerecord-sinatra-lesson zachfeldman $ irb
irb(main):001:0> require './app'
User=> true
irb(main):002:0> User
=> User(id: integer, email: string, fname: string, lname: string, birt
hday: datetime, created_at: datetime, updated_at: datetime)
```

Adding a record

- Now that the `User` model is hooked up, you can use the ActiveRecord methods it has inherited to create, read, update, and destroy information in database tables

- For instance, to add a record, use

```
> User.create(email: "zachfeldman@gmail.com", fname: "Zach")
```

- Looking at the above - attributes of the model are fed to the create method in a Hash-like format as an argument

```
{email: "zachfeldman@gmail.com", fname: "Zach"}
```

Looking up a record

- Records can be looked up by primary key (id)

```
User.find(10)
```

- They can also be looked up by specific traits

```
User.where(fname: "Zach", email: "zach@nycda.com")
```

- The `where` method will return an array of possible results, `find` will always return one User instance

Updating a Record

- You first need an instance of the record to update, which can be found using `.find` or `.where`

- You can use the update_attributes method and pass it a Hash-like list of values to update:

```
> User.find(10).update_attributes(fname: “Zarch”, email: “zarchfieldmon@gm.com”)
```

Updating a Record

- Another option is to save the instance of the record that you're working on in a variable

```
my_user = User.find(10)
```

- Make your changes to the instance that you're working on

```
my_user.fname = "Zach"
```

- When you're done, call the `save` method on the instance to write the information to the database

```
my_user.save
```

Query Strings

- A query string is data sent in the URL using the following syntax:

  https://maps.google.com/maps**?q=fueled**

- To add more things to the query string, we can use the & operator:

  https://maps.google.com/maps?q=fueled**&ip=33.32.323.32**

Sinatra: The `params` hash

- The `params` hash is a Ruby hash object Sinatra uses to store any incoming data to the route
- This can include submitted form data or **query strings!**
- Try this code:

```
get '/sup' do
   puts "THESE ARE MY PARAMS"
   puts params.inspect
end
```

- Then try hitting the url /sup?hi=you and looking in the terminal at the Sinatra server logs. CMD + F for THESE ARE MY PARAMS

Sinatra: The `params` hash

- The parameters inside the `params` hash are accessed just like any other Ruby hash, using the [] syntax:

```
puts params[:hi]
> "you"
```

Sinatra: Instance Variables

- An instance variable as it pertains to Sinatra is just a variable set in a route which you can use in a view

```
#app.rb
get ‘/’ do
  @user = User.find(1)
end

#home.erb
User email: <%= @user.email %>
```

Sinatra: Instance Variables

- You could set an instance variable equal to something that comes in as a parameter in the `params` hash to use it in a view:

```
#app.rb
get ‘/’ do
  @q = params[:q]
end

#home.erb
Your query was: <%= @q %>
```

Using ActiveRecord with Sinatra

- Exercise: First, hookup ActiveRecord to a brand new Sinatra app.
    - Then, have the app's '/' route create a new user whenever the page is hit
    - Assign that new user to `@user` by looking it up in the database (hint: use `User.last`)
    - Display that user's information inside of your homepage view