Quiz 3 - Rails

Full name

sophia mtw denney

Q1 - How do you create a Rails app?

```
➜ rails new app
```

Q2 - How do you start coding a Rails project? Give the right sequence.

1. Coding the views?
2. Coding the controllers?
3. Coding the models?

first routes. then controllers
models
views

Q3 - How do you generate a `Song` model with a `title` and a `year`?

```
➜ rails g model Sont title:string year:integer
```

What are the 2 created files?

song.rb
song_test.rb

What is the **rails** command you should type then?

```
➜ rails db:create  db:migrate
```

Q4 - How do you add a `category` (ex: `"rock"`, `"electro"`, etc..) to your songs table using the correct Rails generator?

```
➜ rails g migration AddCategoryToSongs category:string
```

What is the created file?

timestamp/addCategoryToPost.rb (something along those lines)

What is the **rails** command you should type again?

```
➜ rails db:migrate
```

Q5 - Add a validation on the presence of a song title & crash-test your model in the console

```ruby
# models/song.rb
class Song < ApplicationRecord
  # Add the validation
  validates :song, presence: :true
end
```

Now crash-test your model:

```
➜ rails c
pry>
pry>
pry>
pry>
pry>
pry>
```

Q6 - What is the Rails flow you need to follow again and again? Give the correct order.

1. The Router is routing the HTTP request to "controller#action".
2. The action is getting data from models.
3. Everything starts with an HTTP Request.
4. The action is rendering the view.

3, 1, 2, 4

Q7 - What are the **4 different parts** inside an HTTP request?

1.
2. VERB
   PATH (WWW.GOOGLE.com/search?query=apple)
3. HEADERS
4. BODY

Q8 - Are the HTTP requests in the following 2 routes the same? Why?

```rb
# config/routes.rb
get "/songs" => "songs#index"
post "/songs" => "songs#create"
```

nope the requests are not the same because they are two different types of requests. get is for viewing something and post for changing something.

Q9 - What's the difference between a **GET** and a **POST** request?

get is used for viewing something with out making a change to it
post is implemented when wanting to change something (#create)

Q10 - Complete the controller code using the correct `params` key?

HTTP request:

```
GET /search?query=thriller
```

Routing:

```rb
# config/routes.rb
get "/search" => "songs#search"
```

Controller:

```rb
class SongsController < ApplicationController
  def search
    # TODO
    @songs = Song.find(:id)
  end
end
```

Q11 - Complete the controller code using the correct `params` key?

HTTP request:

```
GET /songs/named/thriller
```

Routing:

```
# config/routes.rb
get "/songs/named/:name" => "songs#search"
```

Controller:

```
class SongsController < ApplicationController
 def search
   # TODO
   @songs = Song.where(title: params[:name])
 end
end
```

Q12 - What are the 7 CRUD routes generated by the `resources` method in Rails?

```
# config/routes.rb
# TODO: Give us the details of the 7 routes generated with:
resources :songs
# HINT: HTTP-verb "url" => "controller#action"
index
new
create
show
edit
update
destroy


```

Q13 - How do you print your routes and their URL prefix helpers?

```
➜ rails routes
```

Q14 - How do you generate a controller for your songs?

```
➜ rails g controller songs
```

Q15 - Implement the **R**ead actions in your songs controller?

```
class SongsController < ApplicationController

  def index
    @songs = Song.all
  end

  def show
    @song = Song.new
  end

end
```

Q16 - What are the 2 requests needed to create a new song? Implement the `songs#new` and `songs#create` actions.

```ruby
class SongsController < ApplicationController
 def new
   @song = Song.new
 end

 def create

  @song = Song.new(song_params)
  if @song.save
     redirect_to _song_path(@song)
  else
  render :new
  end

 end

 private

 def song_params
   params.require(:song).permit(:title, :year)
 end
end
```

Q17 - Why do we have to filter parameters using "strong params" in the controller?

to make clear which params are permitted. we specify in the controller which of the params are validated in a way

Q18 - **Hard question**: What is the HTML generated?

Imagine that:

```ruby
@song = Song.new
```

Now what is the HTML code generated by:

```
<%= form_for @song do |f| %>
  <%= f.text_field :title %>
  <%= f.submit %>
<% end %>
```

Fill the blanks:

```
<form action="songs/new" method="post">
  <input type="text" name="title      " value="year       ">
  <input type="submit" value="Create song">
</form>
```

Q19 - **Hard question**: What is the HTML generated?

Imagine that:

```
@song # => <#Song: id: 18, title: "Hey jude", year: 1968, category: "rock">
```

Now what is the HTML code generated by:

```
<%= form_for @song do |f| %>
  <%= f.text_field :title %>
  <%= f.submit %>
<% end %>
```

Fill the blanks:

```
<form action="/update    " method="patch">
  <input type="text" name="hey jude " value="1968       ">
  <input type="submit" value="Update song">
</form>
```

Adding a 2nd model

Q20 - Now you want to add reviews to your app. Here are some constraints:

- We don't want our visitors to destroy or update reviews, just to create ones.
- We don't want a separate index page to list all reviews or a show page to display each review. Instead, we want to display reviews on the show page of each song, for better UX.

Step #1: Model

Generate your `Review` model in the terminal. It should have only a `content:string` and a `song:references` (= the foreign key).

```
➜ rails g model Review content:string song:regferences
```

Run the migration

```
➜ rails db:migrate
```

Add validation/associations

- Add a validation for the presence of a content
- Add associations between `Review` and `Song`

```
class Song < ApplicationRecord
  has_many :reviews
end
```

```
class Review < ApplicationRecord
  belongs_to :song
  validates :content, presence: true
end
```

Step #2: Routing/Controller

Generate the reviews controller

```
➜ rails g controller reviews
```

Add the **necessary** routes (don't forget **we don't want the 7 CRUD actions for reviews**)

```
# config/routes.rb
resources :songs do
 # TODO
 resources :songs, only [:new, :create]
end
```

Now code your controller:

```
class ReviewsController < ApplicationController
 before_action :set_song

 def new
  @review = Review.new
 end

 def create
  @review = Review.new(review_params)
  @review.song = @song  # puts song's PK as FK in review record
  if @review.save
     redicrect_to_ song_path(@song)
  else
     render:new
 end

 private

 def set_song
   @song = Song.find(params[:song_id])
 end

 def review_params
   params.require(:review).permit(:content)
 end
end
```

Step #3: Views

Add a song's reviews on its show page:

```
<h1><%= @song.title %></h1>
<p><%= @song.year %></p>
<p><%= @song.category %></p>
<h2>Here are the reviews for this song:</h2>
<% @song.reviews.each do |review| %>
      <%= review.content %>
<% end %>


```