Part 1 - A New Rails Application

This challenge assumes that you know how to use git for version control. If you need a review visit the Git Overview notes.

1. Create a news Rails project called ‘simple_store’.

2. Make your first commit. Your project is also a Git repo. You can use VS Code to commit or the command line: git commit -m "Your commit message."

3. Use the rails command line tool to generate a Model called *Product*. A product is defined as having:

- title (string)
- description (text)
- price (decimal)
- stock_quantity (integer)

4. Add your next git commit.

5. Migrate your database and then start your Rails server for this project.

6. Add validation to the Product model to ensure that the product always includes a title, price and stock_quantity. Ensure that the validations are working by trying to add an invalid product from your rails console or seed script.

7. Add your next git commit.

8. Add the faker gem to your Gemfile and run "bundle install" from the project root.

9. Write a short db/seeds.rb file to populate the products table with 676 products. Fake the required product properties using the faker gem. For now it's only necessary to fake the title, price (using Faker::Commerce)

and stock quantity (using Faker::Number).

10. Run your seed script and then make your next git commit. Remember that the seed script needs to be executed using: `rails db:seed`

11. Use the rails command line controller generator to create a Products controller with an index action/view.

12. Add your next git commit.

13. Update the Product controllers index action to load all products into an instance variable.

14. Update the associated index view file to loop through and display all products.

15. Update the routes such that the Products controller's index action is trigger from a GET request to /products

16. Add your next git commit.

17. Add a product show action/view and route to display individual products.

18. Update the index view so that each product name in your list of products links to its show page.

19. Add your next git commit.

Part 2 - Loading Data From a CSV

Ensure that you are incrementally committing to git throughout part 2.

1. Generate a new model called Category with a single string property called "name".

2. To avoid foreign key issues you will first need to delete all the data in your products table. Jump into a rails console from the command line using "`rails c`" and then run "`Product.destroy_all`".

3. Use a migration to add a foreign key reference from your Product model to a category:

```
rails g migration add_category_to_products category:references

rails db:migrate
```

4. Add the associations (belongs_to / has_many) to the Category and Product models.

5. Create a `products.csv` file in your db folder and fill it with the contents found here.

6. Add `require "csv"` to the top of your `db/seeds.rb` file.

7. Rewrite your `db/seeds.rb` file such that it now pulls in product and category data from your `products.csv` file. See appendix below for some assistance on how to pull data from a CSV file using csv library.

8. Update your product index and show views such that all products are display with the name of their associated category.

9. Be sure to avoid a N+1 issue by loading your categories along side your products in the products controller's index action:

```
@products = Product.includes(:category).all
```

Submission

Submit a zip of the entire simple_store folder to the appropriate dropbox.

Appendix A - Loading Data from a CSV in your seed script.

A few things to consider when writing your new db/seeds.rb script:

1. Start the script by clearing our the products and categories tables.

```
Product.destroy_all

Category.destroy_all
```

2. Loop through the rows of a CSV file in your seed script like this:

```
csv_file = Rails.root.join('db/products.csv')
csv_data = File.read(csv_file)

products = CSV.parse(csv_data, headers: true)

# If CSV was created by Excel in Windows you may also need to set an encoding type:
# products = CSV.parse(csv_data, headers: true, encoding: 'iso-8859-1')

products.each do |product|

  # Create categories and products here.

end
```

3. For each product you'll need to first create the associated category. Or, if the category already exists in the database, find it.

```
category = Category.find_or_create_by(name: category_name)

# Where "category_name" is the category name as a string. You will need to get
this from the data returned from the csv library.
```