layout: page

Little Shop Redux

Abstract

In this project you'll use Ruby, Sinatra, and Activerecord to build a site that shows items by categories and analyzes those items.

Learning Goals

ActiveRecord

- Use ActiveRecord migrations to create a normalized database.
- Utilize ActiveRecord methods and relationships to efficiently query the database.
- Use beginner/intermediate ActiveRecord queries to calculate and report on information in the database.

User Experience and Conventions

- Use Sinatra and ERB templates to render views to create, read, update, and delete resources using RESTful routes and appropriate HTTP verbs.
- Use Sinatra and ERB templates to display a dashboard not related to a specific resource saved in the database.
- Use HTML, CSS, and FlexBox to create a user experience that allows users to comfortably navigate a site.

Code Organization/Quality

- Organize code using best practices (use POROs when appropriate, avoid long methods, etc.).
- Create methods using ActiveRecord on the appropriate class.
- No logic should live in the views.

Testing

- Use RSpec to drive development at the feature and model levels.

Working Collaboratively

- Use Git Merge Workflow and GitHub to work collaboratively, develop in smaller groups, and resolve merge conflicts

Getting Started

Define the Relationship with Your Group

DTR with your group here. One group member should send a link of the forked gist to your anchor as soon as it's complete.

Clone the Project

1. One team member forks this skeleton repository.
2. Add the other team members as collaborators.
3. Add instructors as collaborators.

Requirements

This project must use:

- Sinatra
- PostgreSQL
- ActiveRecord

This base repo has already configured those three for you.

You'll want to set up the DatabaseCleaner gem in order to have a clean database each time you run tests. Follow the instructions for setting up the gem. Due to a bug in the most recent version of the gem, you'll need to use this line when you set the strategy in your test helper file:

```
DatabaseCleaner.strategy = :truncation
```

See the "Resources" section at the bottom of this page for additional helpful documentation.

Restrictions

The project may not use:

- `Rails`

Usability

The application has been styled.
The application uses a balanced, considered color scheme.
The application implements a font (that is not the default font).
The application utilizes a nav bar.
The style shows evidence of intentional layout.
Space and text is balanced. White space is used to visually separate content.
The application is easily usable. The user can intuitvely navigate between different portions of the application without manually entering the URL or using the back button on their browser.
The application is minimally responsive.
The application follows the design laid out in these wireframes.

Instructions

Iterations 1-11 must be completed in order to consider the project complete.
Please TDD throughout. Tests are expected for all features and all models.

Base Expectations

Iteration 1 - Merchants

Create full CRUD functionality for a Merchant with the following characteristics:

- name - must be present

Once you have the `Merchant` model started, finish it off by creating validations for the `Merchant` attributes.

You can use ActiveRecord's validations feature to make sure no record is saved without having all attributes present.

Be sure to have a test for each individual validation.

At the end of this iteration, you should be able to view an index of all merchants, view a page for a single merchant, create a merchant, edit a merchant, and delete a merchant from both the index and the show pages

Iteration 2 - Seed Merchants

Update the `seeds` file in your /db directory to parse the `merchants.csv`. When you run rake db:seed your development database should be populated with the information from the merchants.csv file. Your index should include a total of 475 merchants.

Iteration 3 - Items

Create full CRUD functionality for an Item with the following characteristics:

- title
- description
- price
- image

Once you have the `Item` model started, finish it off by creating validations for the `Item` attributes.

- All the attributes must be present.

You can use ActiveRecord's validations feature to make sure no record is saved without having all attributes present.

Be sure to have a test for each individual validation.

At the end of this iteration, you should be able to view an index of all items, view a page for a single item, create an item, edit an item, and delete an item from both the index and the show

pages.

- When creating an item, there should be a dropdown of all merchants to select from.
- You will want to utilize a default image for seeds.
- An image should be a string referencing a url to a photo.

Iteration 4 - Seed Items

Update the `seeds` file in your /db directory to parse the `items.csv`. When you run rake db:seed your development database should be populated with the information from the items.csv file. Be sure to not duplicate data when seeding.

Iteration 5 - Invoices

Create full RUD functionality (No Create) for an Invoice with the following characteristics:

The following attributes must be present

- merchant_id
- status

Once you have the `Invoice` model started, finish it off by creating validations for the `Invoice` attributes.

You can use ActiveRecord's validations feature to make sure no record is saved without having all attributes present.

Be sure to have a test for each individual validation.

At the end of this iteration, you should be able to view an index of all invoices, view a page for a single invoice, edit an invoice, and delete an invoice from both the index and the show pages.

Iteration 6 - Seed Invoices

Update the `seeds` file in your /db directory to parse the `invoices.csv`. When you run rake db:seed your development database should be populated with the information from the `invoices.csv` file. Your index should include a total of 4985 invoices.

Iteration 7 - Relating Items to Invoices

Up until now, you've been working with one-to-many relationships. However, an invoice doesn't mean too much if you don't know which items are assoicated with it.

Build out an InvoiceItem model with the following attributes:

- item_id
- invoice_id
- quantity
- unit_price

We don't want to see a view for this InvoiceItem joins table. However, we do want to see this information SOMEWHERE. Add the following functionality to the Invoice show.

- For each item, list the quantity and unit_price
- List the total price for the invoice

Iteration 8 - Seed Invoice Items

Update the `seeds` file in your /db directory to parse the `invoice_items.csv`. When you run rake db:seed your development database should be populated with the information from the `invoice_items.csv` file. Your index should include a total of 21830 invoice_items.

Iteration 9 - Items Dashboard

Create an items dashboard route. When you visit `/items-dashboard` you should be shown a page with the following information:

- Total count of items
- Average price per item
- Most recently created item
- Oldest item

Iteration 10 - Merchants Dashboard

Create a merchants dashboard route. When you visit `/merchants-dashboard` users should be shown a page with the following information:

Broken Down by Each Merchant

- Total number of items for this merchant
- Average price of item for this merchant
- Total price for all items for this merchant
- Merchant with the most items and that merchant's information
- Merchant with the highest priced item and that merchant's information

Iteration 11 - Invoices Dashboard

Create a invoices dashboard route. When you visit `/invoices-dashboard` users should be shown a page with the following information:

- Each Invoice status as a percent of total invoices
- Invoice with the highest associated unit_price
- Invoice with the lowest associated unit_price
- Invoice with the highest assoicated quantity
- Invoice with the lowest assoicated quantity

Extensions

- Use Google Charts to display information on one or more of your dashboards.
- Read about JSON. Create an endpoint at `api/v1/items/:id` that responds to requests with JSON instead of HTML.
- Deploy your application to Heroku. You'll need to create fixtures of your data to meet the constraints of the free account on Heroku.

Evaluation Process

For the evaluation we'll work through the expectations above and look at the following criteria:

1. Feature Completeness

- Exceeds Expectations: All features are correctly implemented along with two extensions
- Meets Expectations: All features defined in the assignment are correctly implemented
- Below Expectations: There are one or two features missing or incorrectly implemented

2. Views

- Exceeds Expectations: Views show logical refactoring into layout(s), partials and helpers, with no logic present
- Meets Expectations: Views make use of layout(s)
- Below Expectations: Views show weak understanding of `erb` and 'HTML'

3. Controller

- Exceeds Expectations: Controller show significant effort to push logic down the stack
- Meets Expectations: Controller is generally well organized with three or fewer methods needing refactoring
- Below Expectations: There are four to seven controller methods that should have been refactored

4. Models

- Exceeds Expectations: Models show excellent organization, refactoring, and appropriate use of ActiveRecord features
- Meets Expectations: Models show an effort to push logic down the stack, but need more internal refactoring
- Below Expectations: Models are somewhat messy and/or make poor use of ActiveRecord features

5. ActiveRecord

- Exceeds Expectations: Best choice ActiveRecord methods are used to solve each problem
- Meets Expectations: ActiveRecord is utilized wherever it can be. There is no Ruby where there should be ActiveRecord
- Below Expectations: Ruby is used to programatically solve problems where ActiveRecord could be used

6. Testing

- Exceeds Expectations: Project has a running test suite that covers all functionality, exercises the application at multiple levels, and covers edge cases
- Meets Expectations: Project has a running test suite that tests at multiple levels
- Below Expectations: Project has sporadic use of tests

7. Usability

- Exceeds Expectations: Project is highly usable and ready to deploy to customers
- Meets Expectations: Project is usable, but needs more polish or navigation before it'd be customer-ready
- Below Expectations: Project needs more attention to the User Experience, but works

8. Workflow

- Exceeds Expectations: Excellent use of branches, pull requests, code review and a project management tool.
- Meets Expectations: Good use of branches, pull requests, and a project-management tool.
- Below Expectations: Sporadic use of branches, pull requests, and/or project-management tool.