In this chapter, you'll see:

- Reviewing Rails concepts: model, view, controller, configuration, testing, and deployment
- Documenting what we've done

CHAPTER 18

Depot Retrospective

Congratulations! By making it this far, you've obtained a solid understanding of the basics of every Rails application. There's much more to learn, which we'll pick back up again in Part III. For now, relax, and let's recap what you've seen in Part II.

Rails Concepts

In Chapter 3, The Architecture of Rails Applications, on page 37, we introduced models, views, and controllers. Now let's see how we applied each of these concepts in the Depot application. Then let's explore how we used configuration, testing, and deployment.

Model

Models are where all of the persistent data retained by your application is managed. In developing the Depot application, we created five models: Cart, LineItem, Order, Product, SupportRequest, and User.

By default, all models have id, created_at, and updated_at attributes. To our models, we added attributes of type string (examples: title, name), integer (quantity), text (description, address), and decimal (price), as well as foreign keys (product_id, cart_id). We even created a virtual attribute that's never stored in the database—namely, a password.

We created has_many and belongs_to relationships that we can use to navigate among our model objects, such as from Carts to LineItems to Products.

We employed migrations to update the databases, not only to introduce new schema information but also to modify existing data. We demonstrated that they can be applied in a fully reversible manner.

The models we created were not merely passive receptacles for our data. For starters, they actively validate the data, preventing errors from propagating. We created validations for presence, inclusion, numericality, range, uniqueness, format, and confirmation (and length, too, if you completed the exercises). We created custom validations for ensuring that deleted products aren't referenced by any line item. We used an Active Record hook to ensure that an administrator always remains and used a transaction to roll back incomplete updates on failure.

We also created logic to add a product to a cart, add all line items from a cart to an order, encrypt and authenticate a password, and compute various totals.

Finally, we created a default sort order for products for display purposes.

View

Views control the way our application presents itself to the external world. By default, Rails scaffolding provides `edit`, `index`, `new`, and `show`, as well as a partial named `form` that's shared between `edit` and `new`. We modified a number of these and created new partials for carts and line items.

In addition to the model-backed resource views, we created entirely new views for `admin`, `sessions`, and the `store` itself.

We updated an overall layout to establish a common look and feel for the entire site. We updated in a style sheet. We made use of partials and added JavaScript to takes advantage of HotWired and WebSocket technologies to make our website more interactive.

We localized the customer views for display in both English and Spanish.

Not all of the views were designed for browsers: we created views for email too, and those views were able to share partials for displaying line items.

Controller

By the time we were done, we created eight controllers: one each for the six models and the three additional ones to support the views for `admin`, `sessions`, and the `store` itself.

These controllers interacted with the models in a number of ways, from finding and fetching data and putting it into instance variables to updating models and saving data entered via forms. When done, we either redirected to another action or rendered a view.

We limited the set of permitted parameters on the line item controller.

We created callback actions that were run before selected actions to find the cart, set the language, and authorize requests. We placed logic common to a number of controllers into a concern—namely, the CurrentCart module.

We managed sessions, keeping track of the logged-in user (for administrators) and carts (for customers). We kept track of the current locale used for internationalization of our output. We captured errors, logged them, and informed the user via notices.

We employed fragment caching on the storefront.

We also sent confirmation emails on receipt of an order.

Configuration

Conventions keep to a minimum the amount of configuration required for a Rails application, but we did do a bit of customization.

We modified our database configuration to use MySQL in production.

We defined routes for our resources, admin and session controllers, and the *root* of our website—namely, our storefront.

We created an initializer for i18n purposes and updated the locales information for both English (en) and Spanish (es).

We created seed data for our database.

We created a Docker configuration for deployment, including the definition of a secret.

Testing

We maintained and enhanced tests throughout.

We employed unit tests to validation methods. We also tested increasing the quantity on a given line item.

Rails provided basic tests for all our scaffolded controllers, which we maintained as we made changes. We added tests along the way for things such as Ajax and ensuring that a cart has items before we create an order.

We used fixtures to provide test data to fuel our tests.

We created an integration test to test an end-to-end scenario involving a user adding product to a cart, entering an order, and receiving a confirmation email.

Deployment

We deployed our application to a production-quality web server (nginx) using a production-quality database server (PostGreSQL). Along the way, we installed and configured Phusion Passenger to run our application, Bundler to track dependencies, and Git to configuration manage our code. Docker compose was employed to orchestrate updating the deployed web server in production from our development machine.

We made use of test and production environments to prevent our experimentation during development from affecting production. Our development environment made use of the lightweight SQLite database server and web server, Puma. Our tests were run in a controlled environment with test data provided by fixtures.

Documenting What We've Done

To complete our retrospective, let's see how much code we've written. There's a Rails command for that too:

```
% bin/rails stats
+----------------------+--------+--------+---------+---------+-----+-------+
| Name                 |  Lines |    LOC | Classes | Methods | M/C | LOC/M |
+----------------------+--------+--------+---------+---------+-----+-------+
| Controllers          |    614 |    410 |      10 |      59 |   5 |     4 |
| Helpers              |     18 |     18 |       0 |       0 |   0 |     0 |
| Jobs                 |     18 |      8 |       2 |       1 |   0 |     6 |
| Models               |    165 |    113 |       8 |       7 |   0 |    14 |
| Mailers              |     51 |     22 |       3 |       3 |   1 |     5 |
| Mailboxes            |     22 |     16 |       2 |       1 |   0 |    14 |
| Channels             |     19 |     15 |       3 |       2 |   0 |     5 |
| Views                |    938 |    758 |       0 |       0 |   0 |     0 |
| Stylesheets          |     79 |     68 |       0 |       0 |   0 |     0 |
| JavaScript           |     88 |     51 |       0 |       0 |   0 |     0 |
| Libraries            |     34 |     33 |       1 |       1 |   1 |    31 |
| Controller tests     |    395 |    280 |       8 |      45 |   5 |     4 |
| Helper tests         |      0 |      0 |       0 |       0 |   0 |     0 |
| Job tests            |      7 |      3 |       1 |       0 |   0 |     0 |
| Model tests          |    139 |     95 |       6 |       9 |   1 |     8 |
| Mailer tests         |     63 |     44 |       4 |       6 |   1 |     5 |
| Mailbox tests        |     58 |     32 |       1 |       2 |   2 |    14 |
| Channel tests        |     19 |      6 |       2 |       0 |   0 |     0 |
| Integration tests    |      0 |      0 |       0 |       0 |   0 |     0 |
| System tests         |    184 |    134 |       3 |      10 |   3 |    11 |
| Model specs          |     58 |     36 |       0 |       0 |   0 |     0 |
+----------------------+--------+--------+---------+---------+-----+-------+
| Total                |   2969 |   2142 |      54 |     146 |   2 |    12 |
+----------------------+--------+--------+---------+---------+-----+-------+
  Code LOC: 1512     Test LOC: 630     Code to Test Ratio: 1:0.4
```

Think about it: you've accomplished a lot and with not all that much code. And much of it was generated for you. This is the magic of Rails.