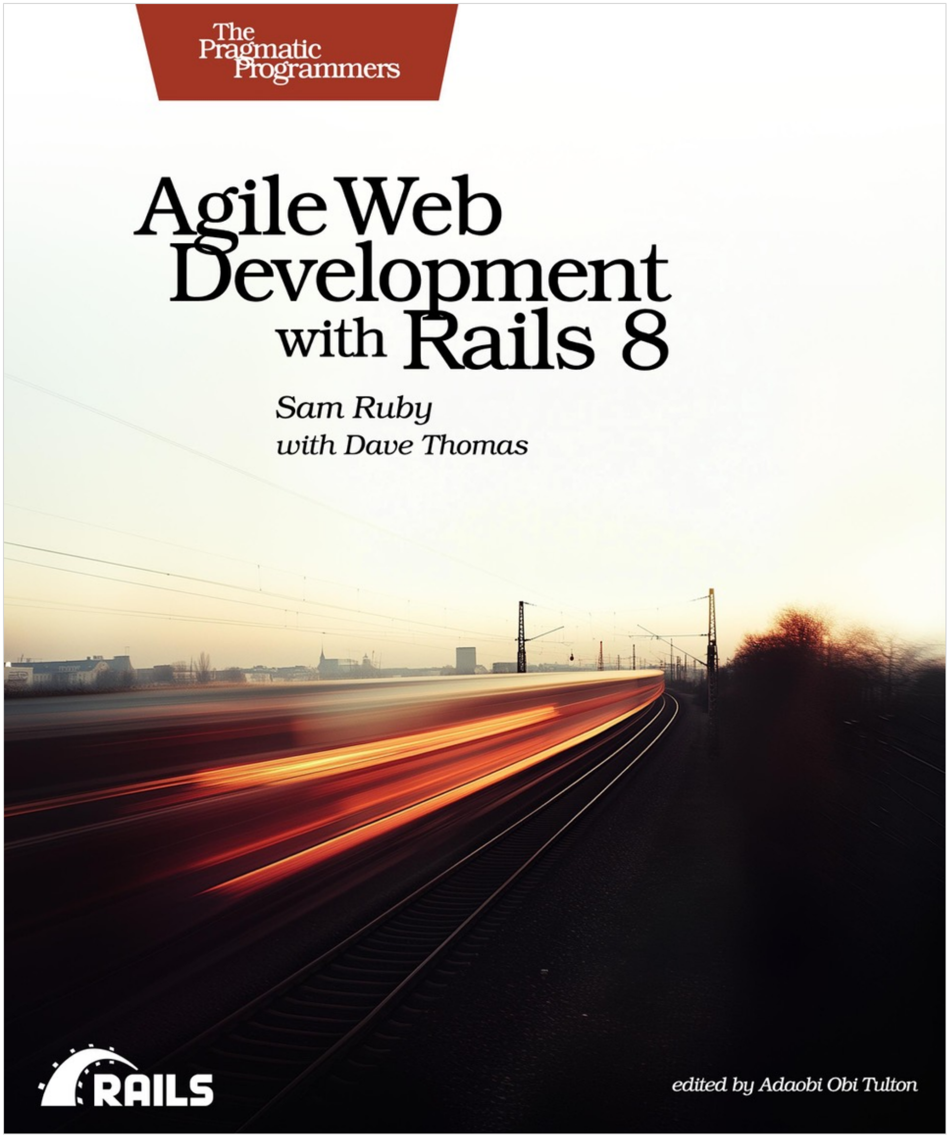

The Pragmatic Programmers
Agile Web Development with Rails 8
Sam Ruby
with Dave Thomas
RAILS
edited by Adaobi Obi Tulton

## Iteration A1: Creating the Product Maintenance Application

At the heart of the Depot application is a database. Getting this installed and configured and tested before proceeding will prevent a lot of headaches. If you're not certain about what you want, take the defaults, and it will go easily. If you know what you want, Rails makes it easy for you to describe your configuration.

For this project, let's make use of the Tailwind CSS[1] framework, which enables you to make pretty websites without authoring any CSS. We'll do so by specifying an additional option when we create our application, and as you'll shortly see it will also affect how we start our server during development.

### Creating a Rails Application

In Creating a New Application, on page ?, you saw how to create a new Rails application. We'll do the same thing here. Go to a command prompt and type rails new followed by the name of our project, and then add the option to make use of the Tailwind CSS framework. Here, our project is called depot, so make sure you're not inside an existing application directory, and type this:

```
work> rails new depot --css tailwind
```

We see a bunch of output scroll by. When it has finished, we find that a new directory, depot, has been created. That's where we'll be doing our work:

```
work> cd depot
depot> ls -p
Dockerfile      README.md       config/         log/            test/
Gemfile         Rakefile        config.ru       public/         tmp/
Gemfile.lock    app/            db/             script/         vendor/
Procfile.dev    bin/            lib/            storage/
```

Of course, Windows users need to use dir /w instead of ls -p.

### Creating the Database

For this application, we'll use the open source SQLite database (which you'll need if you're following along with the code). We're using SQLite version 3 here.

SQLite 3 is the default database for Rails development and was installed along with Rails in Chapter 1, Installing Rails, on page ?. With SQLite 3, no steps are required to create a database, and we have no special user accounts or

1. https://tailwindcss.com/

passwords to deal with. So now you get to experience one of the benefits of going with the flow (or, convention over configuration, as the Rails folks say…ad nauseam).

If it's important to you to use a database server other than SQLite 3, the commands to create the database and grant permissions will be different. You can find some helpful hints in the database configuration section of Configuring Rails Applications in the Ruby on Rails Guides.[2]

## Generating the Scaffold

Back in our initial guess at application data on page ?, we sketched out the basic content of the products table. Now let's turn that into reality. We need to create a database table and a Rails *model* that lets our application use that table, a number of *views* to make up the user interface, and a *controller* to orchestrate the application.

So let's create the model, views, controller, and migration for our products table. With Rails, you can do all that with one command by asking Rails to generate a *scaffold* for a given model. Note that on the command line that follows, we use the singular form, Product. In Rails, a model is automatically mapped to a database table whose name is the plural form of the model's class. In our case, we ask for a model called Product, so Rails associates it with the table called products. (And how will it find that table? The development entry in config/database.yml tells Rails where to look for it. For SQLite 3 users, this'll be a file in the storage directory.)

Note that the command is too wide to fit comfortably on the page. To enter a command on multiple lines, put a backslash as the last character on all but the last line, and you'll be prompted for more input. Windows users need to substitute a caret (^) for the backslash at the end of the first line and a backslash for the forward slash in bin/rails:

```
depot> bin/rails generate scaffold Product \
         title:string description:text image:attachment price:decimal
  invoke  active_record
  create    db/migrate/20241021000001_create_products.rb
  create    app/models/product.rb
  invoke    test_unit
  create      test/models/product_test.rb
  create      test/fixtures/products.yml
  invoke  resource_route
   route    resources :products
  invoke  scaffold_controller
```

2. http://guides.rubyonrails.org/configuring.html#configuring-a-database

```
create    app/controllers/products_controller.rb
invoke    tailwindcss
create      app/views/products
create      app/views/products/index.html.erb
create      app/views/products/edit.html.erb
create      app/views/products/show.html.erb
create      app/views/products/new.html.erb
create      app/views/products/_form.html.erb
create      app/views/products/_product.html.erb
invoke    resource_route
invoke    test_unit
create      test/controllers/products_controller_test.rb
create      test/system/products_test.rb
invoke    helper
create      app/helpers/products_helper.rb
invoke      test_unit
invoke    jbuilder
create      app/views/products/index.json.jbuilder
create      app/views/products/show.json.jbuilder
create      app/views/products/_product.json.jbuilder
```

The generator creates a bunch of files. The one we're interested in first is the *migration* one, namely, 20241021000001_create_products.rb.

A migration represents a change we either want to make to a database as a whole or to the data contained within the database, and it's expressed in a source file in database-independent terms. These changes can update both the database schema and the data in the database tables. We apply these migrations to update our database, and we can unapply them to roll our database back. We have a whole section on migrations starting in Chapter 23, Migrations, on page ?. For now, we'll just use them without too much more comment.

The migration has a UTC-based timestamp prefix (20241021000001), a name (create_products), and a file extension (.rb, because it's Ruby code).

The timestamp prefix that you see will be different. In fact, the timestamps used in this book are clearly fictitious. Typically, your timestamps won't be consecutive; instead, they'll reflect the time the migration was created.

## Applying the Migration

Although we've already told Rails about the basic data types of each property, let's refine the definition of the price to have eight digits of significance and two digits after the decimal point:

rails80/depot_a/db/migrate/20241021000001_create_products.rb

```
class CreateProducts < ActiveRecord::Migration[8.0]
  def change
    create_table :products do |t|
      t.string :title
      t.text :description
➤     t.decimal :price, precision: 8, scale: 2

      t.timestamps
    end
  end
end
```

Since we are defined an attachment we need to install the tables that Active Storage uses to track the attachments. This only needs to be done once per database. We do this by running the following command:

```
depot> bin/rails active_storage:install
Copied migration 20241021000002_create_active_storage_tables.active_storage.rb
  from active_storage_attachments
```

Now that we're done with our changes, we need to get Rails to apply this migration to our development database. We do this by using the bin/rails db:migrate command:

```
depot> bin/rails db:migrate
== 20241021000001 CreateProducts: migrating ===================================
-- create_table(:products)
  -> 0.0025s
== 20241021000001 CreateProducts: migrated (0.0025s) ==========================

== 20241021000002 CreateActiveStorageTables: migrating ========================
-- create_table(:active_storage_blobs, {:id=>:primary_key})
  -> 0.0065s
-- create_table(:active_storage_attachments, {:id=>:primary_key})
  -> 0.0270s
-- create_table(:active_storage_variant_records, {:id=>:primary_key})
  -> 0.0113s
== 20241021000002 CreateActiveStorageTables: migrated (0.0451s) ===============
```

And that's it. Rails looks for all the migrations not yet applied to the database and applies them. In our case, the products table is added to the database defined by the development section of the database.yml file, and three tables are created for Active Storage to use.

OK, all the groundwork has been done. We set up our Depot application as a Rails project. We created the development database and configured our application to be able to connect to it. We created a products controller and a Product model and used a migration to create the corresponding products table. And a number of views have been created for us. It's time to see all this in action.

## Seeing the List of Products

With four commands, we've created an application and a database (or a table inside an existing database if you chose something besides SQLite 3) and installed Active Storage. Before we worry too much about what happened behind the scenes here, let's try our shiny new application.

We mentioned previously that using a CSS processor will affect how we start our server during development. This is because things like CSS processors and JavaScript bundlers require a build step. Rather than requiring you to start multiple processes, Rails provides bin/dev, which is a small script that will start everything:

```
depot> bin/dev
08:43:51 web.1  | started with pid 31227
08:43:51 css.1  | started with pid 31228
08:43:52 web.1  | => Booting Puma
08:43:52 web.1  | => Rails 8.0.0.rc1 application starting in development
08:43:52 web.1  | => Run `bin/rails server --help` for more startup options
08:43:52 web.1  | Puma starting in single mode...
08:43:52 web.1  | * Puma version: 6.4.3 (ruby 3.3.5-p100) ("The Eagle of Durango")
08:43:52 web.1  | *  Min threads: 3
08:43:52 web.1  | *  Max threads: 3
08:43:52 web.1  | *  Environment: development
08:43:52 web.1  | *          PID: 31227
08:43:52 web.1  | * Listening on http://127.0.0.1:3000
08:43:52 web.1  | * Listening on http://[::1]:3000
08:43:52 web.1  | Use Ctrl-C to stop
08:43:52 css.1  |
08:43:52 css.1  | Rebuilding...
08:43:53 css.1  |
08:43:53 css.1  | Done in 228ms.
```

Windows users will need to run the command ruby bin/dev.

If you examine that output, in addtion to the lines containing web.1 that show the Rails server starting, you see lines containing css.1 that show the CSS rebuilding. This is all controlled by a file named Procfile.dev:

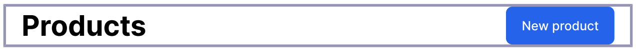
**rails80/depot_a/Procfile.dev**

```
web: bin/rails server
css: bin/rails tailwindcss:watch
```

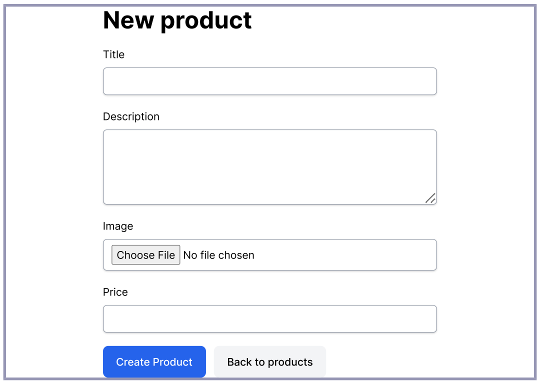
Feel free to modify this file to suit your needs. For example, if you're using a virtual machine, you might need to add -b 0.0.0.0 to the rails server line to accept connections from your host.

As with our demo application on page ?, this command starts a web server on our local host, port 3000. If you get an error saying Address already in use when you try to run the server, that means you already have a Rails server running on your machine. If you've been following along with the examples in the book, that might well be the Hello, World! application from Chapter 4. Find its console and kill the server using Ctrl-C. If you're running on Windows, you might see the prompt Terminate batch job (Y/N)?. If so, respond with y.

Let's connect to our application. Remember, the URL we give to our browser is http://localhost:3000/products, which has both the port number (3000) and the name of the controller in lowercase (products). The application looks like the following screenshot.



That's pretty boring. It's showing us an empty list of products. Let's add some. Click the New Product link. A form should appear, as shown in the next screenshot.



These forms are simply HTML templates, like the ones you created in Hello, Rails!, on page ?. In fact, we can modify them. Let's change the number of rows in the Description field, and limit the acceptable files to select for upload to images:

rails80/depot_a/app/views/products/_form.html.erb
```
<%= form_with(model: product, class: "contents") do |form| %>
  <% if product.errors.any? %>
    <div id="error_explanation"
      class="bg-red-50 text-red-500 px-3 py-2 font-medium rounded-lg mt-3">
      <h2><%= pluralize(product.errors.count, "error") %>
      prohibited this product from being saved:</h2>

      <ul>
        <% product.errors.each do |error| %>
          <li><%= error.full_message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <div class="my-5">
    <%= form.label :title %>
    <%= form.text_field :title, class: "block shadow rounded-md…" %>
  </div>

  <div class="my-5">
    <%= form.label :description %>
➤   <%= form.textarea :description, rows: 10, class: "block shadow…" %>
  </div>

  <div class="my-5">
    <%= form.label :image %>
➤   <%= form.file_field :image, accept: "image/*", class: "block shadow…" %>
  </div>

  <div class="my-5">
    <%= form.label :price %>
    <%= form.text_field :price, class: "block shadow rounded-md…" %>
  </div>

  <div class="inline">
    <%= form.submit class: "rounded-lg py-3 px-5…" %>
  </div>
<% end %>
```

We'll explore this more in Chapter 8, Task C: Catalog Display, on page ?. But for now, we've adjusted two fields to taste, so let's fill it in, as shown in screenshot on page 10 (note the use of HTML tags in the description—this is intentional and will make more sense later).

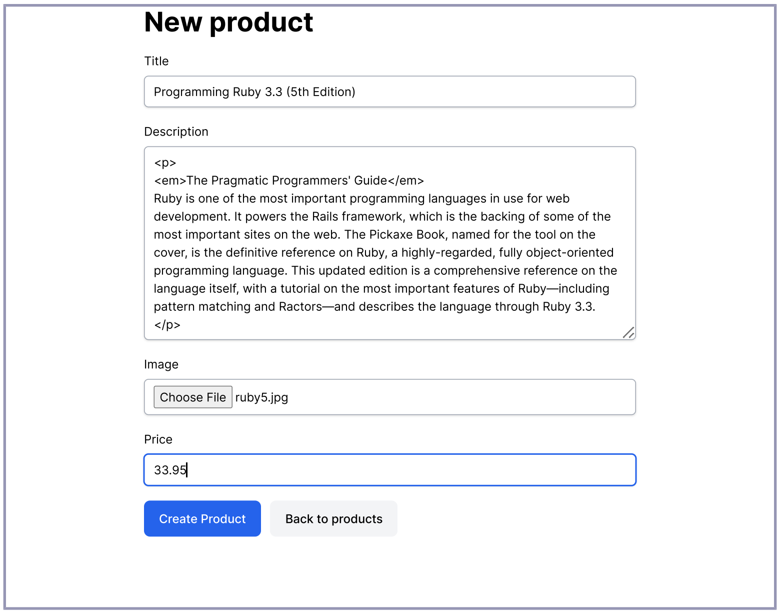
Now we need some files to upload. Create a directory named db/images in your application, and download the images there.[3]

Fill in the fields, select a file, and click the Create button, and you should see that the new product was successfully created. If you now click the Back link, you should see the new product in the list, as shown in the screenshot on page 10.

Perhaps it isn't the prettiest interface, but it works, and we can show it to our client for approval. She can play with the other links (showing details, editing existing products, and so on). We explain to her that this is only a first step—we know it's rough, but we wanted to get her feedback early. (And five commands probably count as early in anyone's book.)

Note that if you've used a database other than SQLite 3, this step may have failed. Check your database.yml file.

3. https://media.pragprog.com/titles/rails80/code/rails80/depot_a/db/images/


New product

Title

Programming Ruby 3.3 (5th Edition)

Description

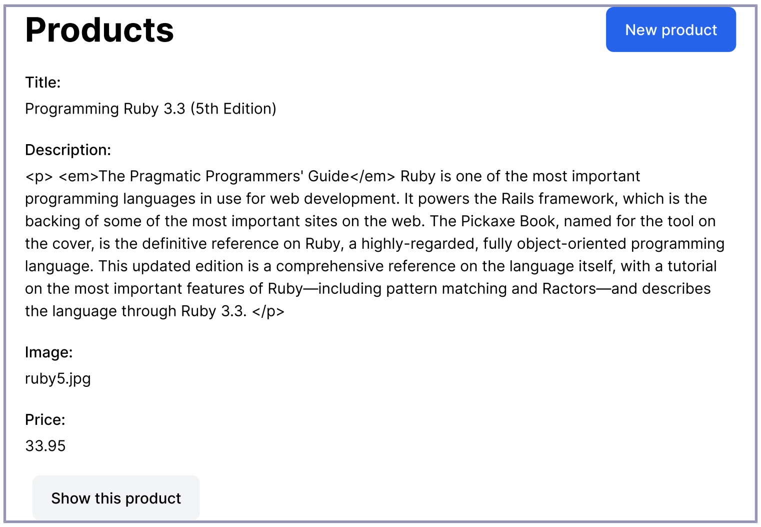
```
<p>
<em>The Pragmatic Programmers' Guide</em>
Ruby is one of the most important programming languages in use for web
development. It powers the Rails framework, which is the backing of some of the
most important sites on the web. The Pickaxe Book, named for the tool on the
cover, is the definitive reference on Ruby, a highly-regarded, fully object-oriented
programming language. This updated edition is a comprehensive reference on the
language itself, with a tutorial on the most important features of Ruby—including
pattern matching and Ractors—and describes the language through Ruby 3.3.
</p>
```

Image

Choose File ruby5.jpg

Price

33.95


Products

Title:
Programming Ruby 3.3 (5th Edition)

Description:
<p> <em>The Pragmatic Programmers' Guide</em> Ruby is one of the most important programming languages in use for web development. It powers the Rails framework, which is the backing of some of the most important sites on the web. The Pickaxe Book, named for the tool on the cover, is the definitive reference on Ruby, a highly-regarded, fully object-oriented programming language. This updated edition is a comprehensive reference on the language itself, with a tutorial on the most important features of Ruby—including pattern matching and Ractors—and describes the language through Ruby 3.3. </p>

Image:
ruby5.jpg

Price:
33.95