Web Engineering: Building an application

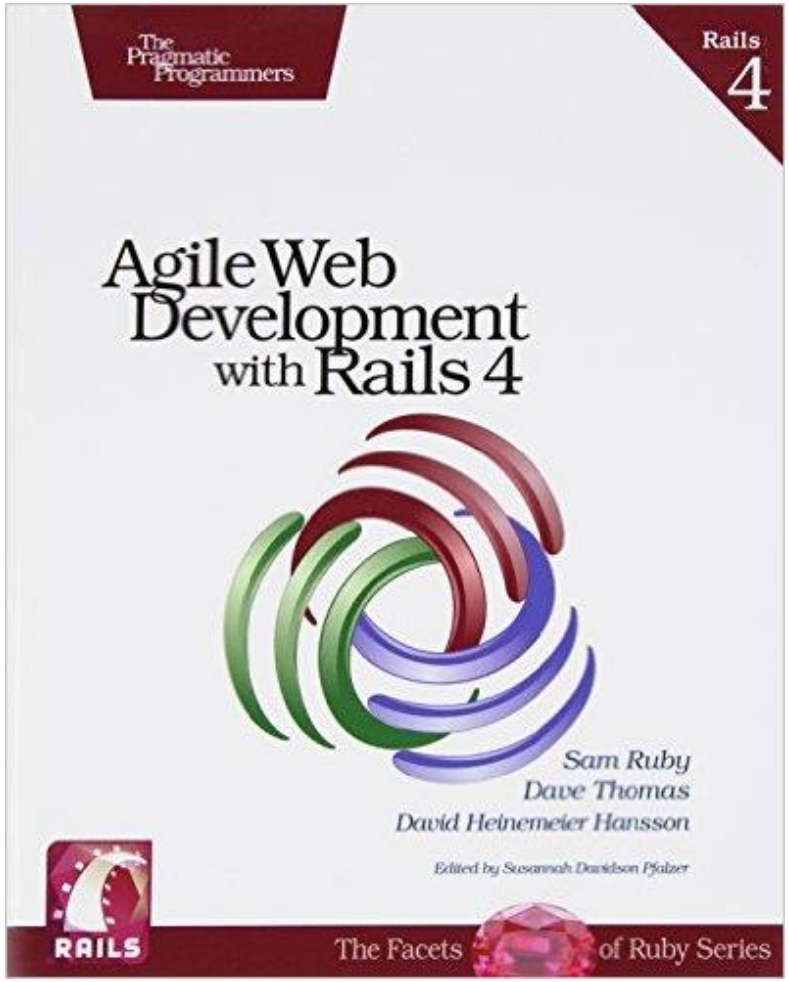
The University of Aizu
Quarter 2, AY 2018

Outline

- Example: the Depot application
  - Incremental development
  - What Depot does
- Creating the application
  - Creating the products maintenance application
  - Making prettier listings

Literature

- Agile Web Development with Rails 4 (1st edition) by Sam Ruby, Dave Thomas and Devid Hansson, The Pragmatic Bookshelf, 2013.
  - Chapters 5 and 6.



The Depot application

- We will be creating a Web-based shopping cart application called ***Depot***.
- Our shopping cart will illustrate how to create:
  - Simple maintenance pages,
  - Link database tables,
  - Handle sessions and,
  - Create forms.

Incremental development

- We will be developing the application incrementally
- We won't attempt to specify everything before we start coding.
- We will be working on a specification to let us start and then immediately create some functionality.
- We will try ideas, gather feedback, and continue with another cycle of mini-design and development.

Incremental development

- We need to use a toolset that does not penalize us for changing our minds.
- We need to be able to make any changes without a bunch of coding or configuration.
- Ruby on Rails is an ideal agile programming environment.

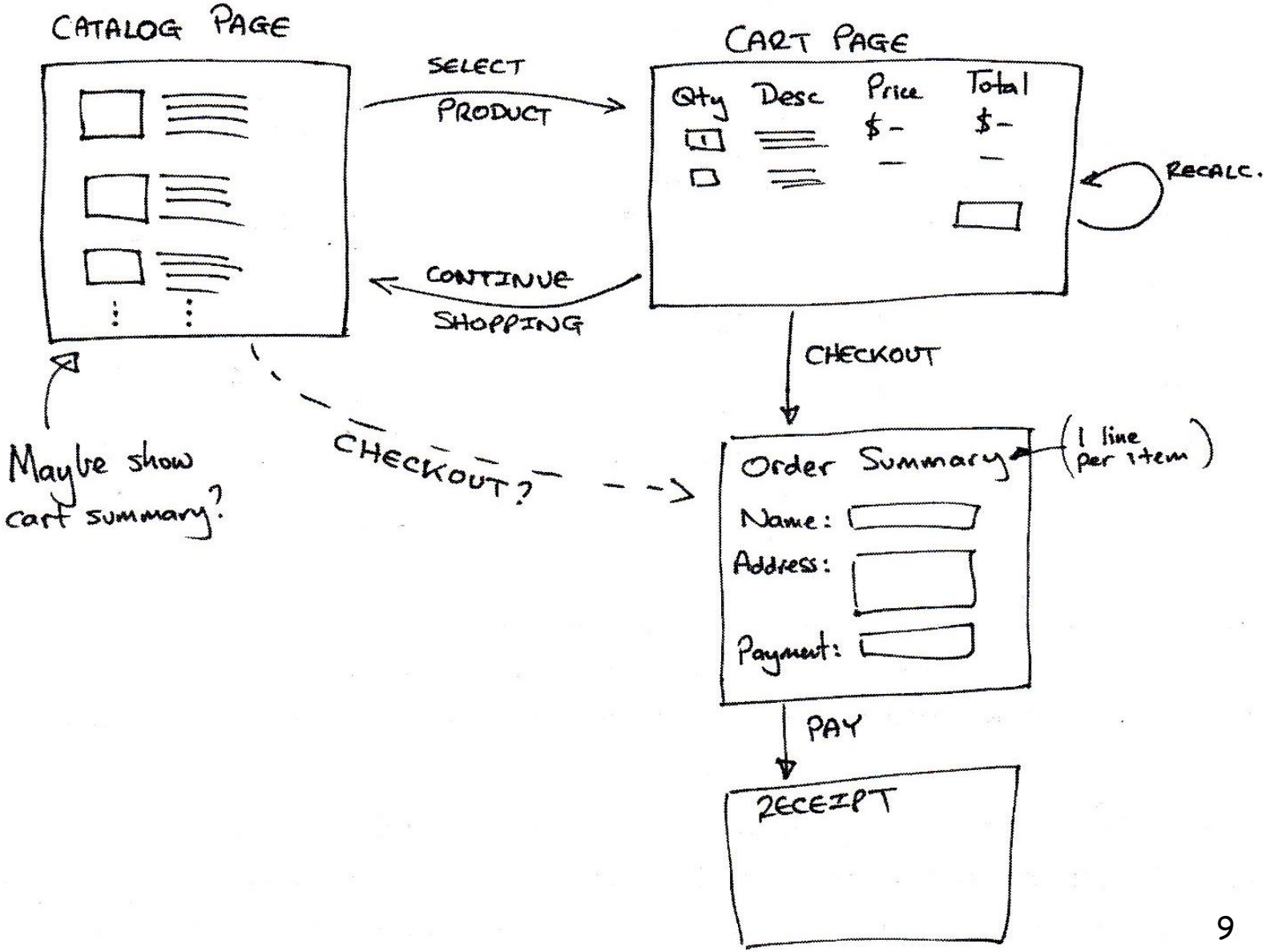
What Depot does

- A use case is a statement about how some entity uses a system.
- Use cases:
  - Actors (roles)
    - Buyer: uses Depot to browse the products we offer to sell, select some to purchase, and provide the information to create the order;
    - Seller: uses Depot to maintain a list of products to sell, to determine the orders that are awaiting shipping, and to mark orders as shipped.

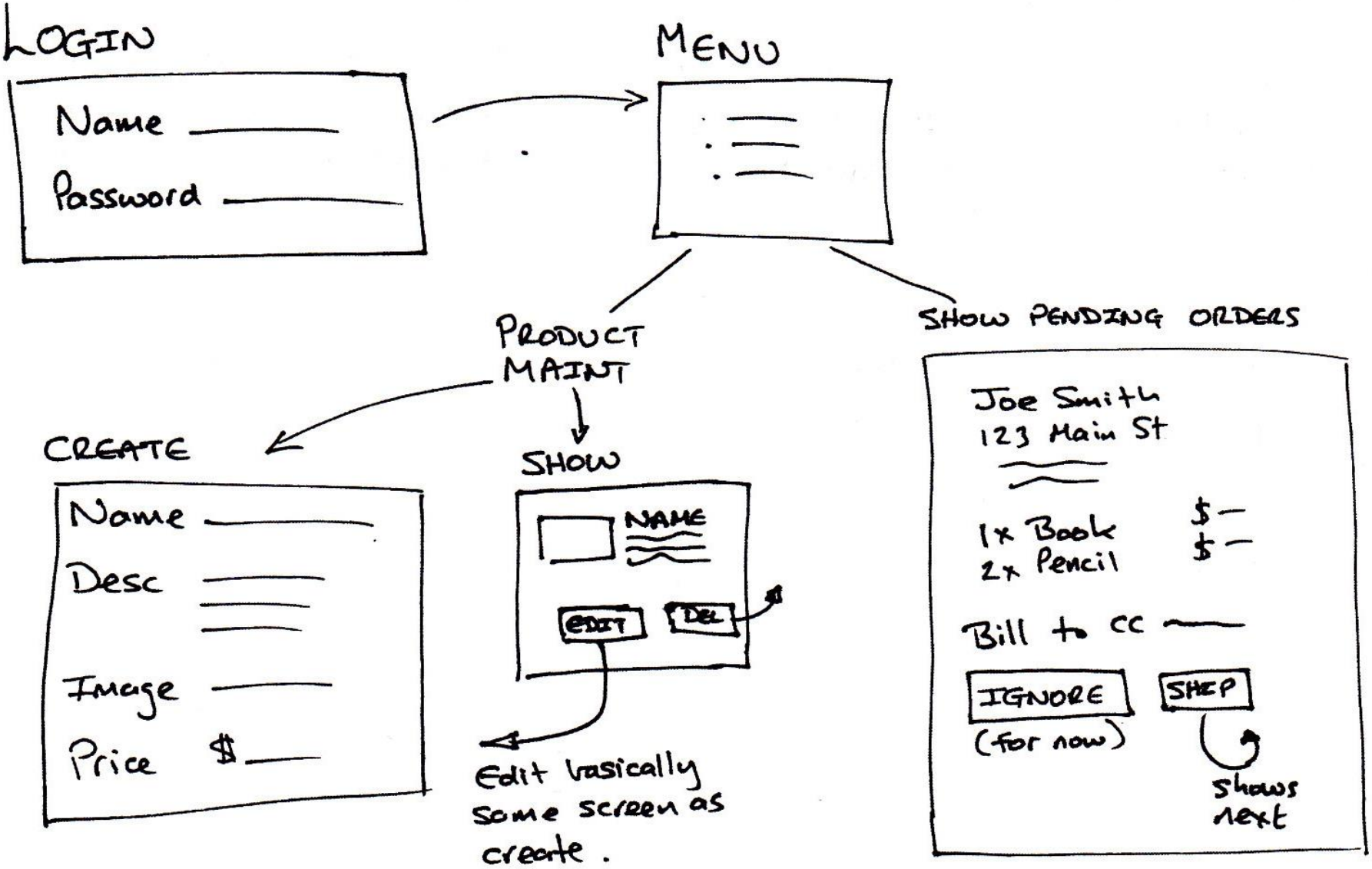
Page flow

- We would like to have an idea of main pages in our applications and to understand how users navigate between them.
- These page flows are incomplete, but they help us focus on what needs doing and know how actors are sequenced.
- We will start with pencil and paper to create page flows.

Flow of buyer pages



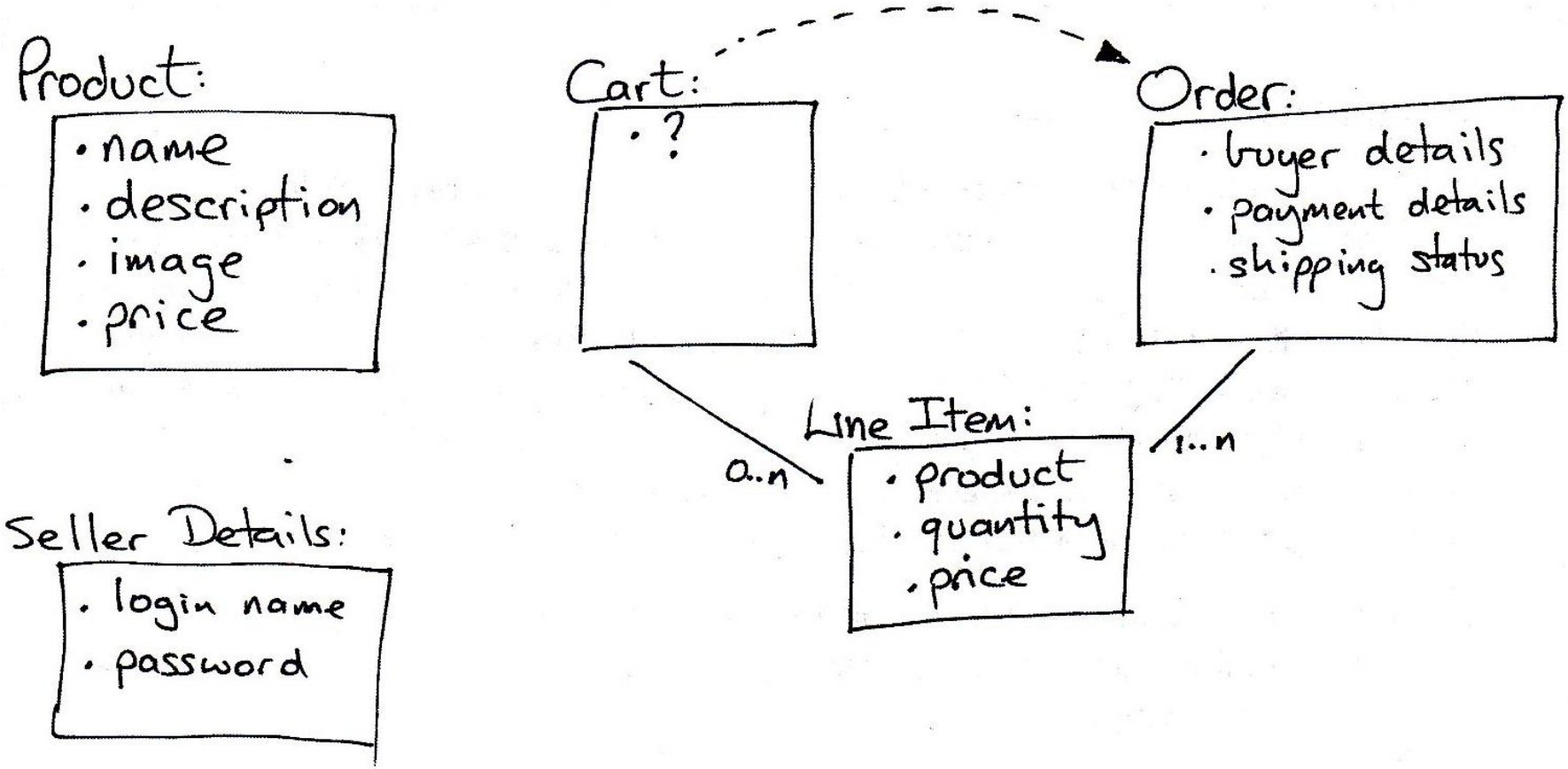
Flow of seller pages



Data

- We need to think about the data we are going to be working with.
- We simply talking about data. At this stage, we do not know whether we will be using a database.

Initial guess at application data



Data analysis

- As the user buys items, we need somewhere to keep the list of products they bought.
  - We use cart for this purpose.
  - On the other hand, we could not find anything meaningful to store in it.
    - To reflect this uncertainty, we put "?" inside the cart box in the diagram.
- What information should go into an order
  - We will refine this later when communicating with the customer and awaiting his/her feedback.
- We have duplicated the product's price in the line item data.

We are ready to create code

- We will be working from our THREE diagrams, but the chances are good that we will throwing them away very quickly!
  - They will become outdated as we gather feedback.
- It is easy to throw something away if you did not spend a long time creating it!
- We will develop this application in small iterations, where small means: "measured in minutes".

Creating a Rails application

- The first step:

  work> rails new depot --skip-bundle

  - Here "depot" is the name of our project.

Generating the Scaffold

- See Slide 12: We need to create
  - the model,
  - Views,
  - Controller, and
  - Migration for our product tables.
- With Rails, all above things can be done with one command generating what is known as a *scaffold* for a given model.

Generating the Scaffold

- In the command below, we asked for a model called *Product*, Rails associated it with the table called *products*.
  - 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 will be the file in the *db* directory.

```
depot> rails generate scaffold Product \
title:string  description:text image_url:string \
price:decimal
```

Generating the Scaffold

- Results of the command from the previous slide are a bunch of files.
- We are interested in the *migration* one
  - 20121130000001_create_products.rb
    - Here: UTC-based timestamp (20121130000001)
    - rb: Ruby program
- A migration represents a change we want to make in a source file in database-independent terms.
- These changes can update both the database schema and the data in the database tables.

Applying migrations

- We will refine the definition of the price to have 8 digits of significance and 2 digits after the decimal point:

Download rails40/depot_a/db/migrate/20121130000001_create_products.rb

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

      t.timestamps
    end
  end
end
```

- Now, we need to apply our changes to our development database (see the next slide).

Applying migrations

- We use rake command:
  - Rake is a reliable assistant.
  - We tell Rake to apply any unapplied migrations to our database.

    ```
    depot > rake db:migrate
    ```

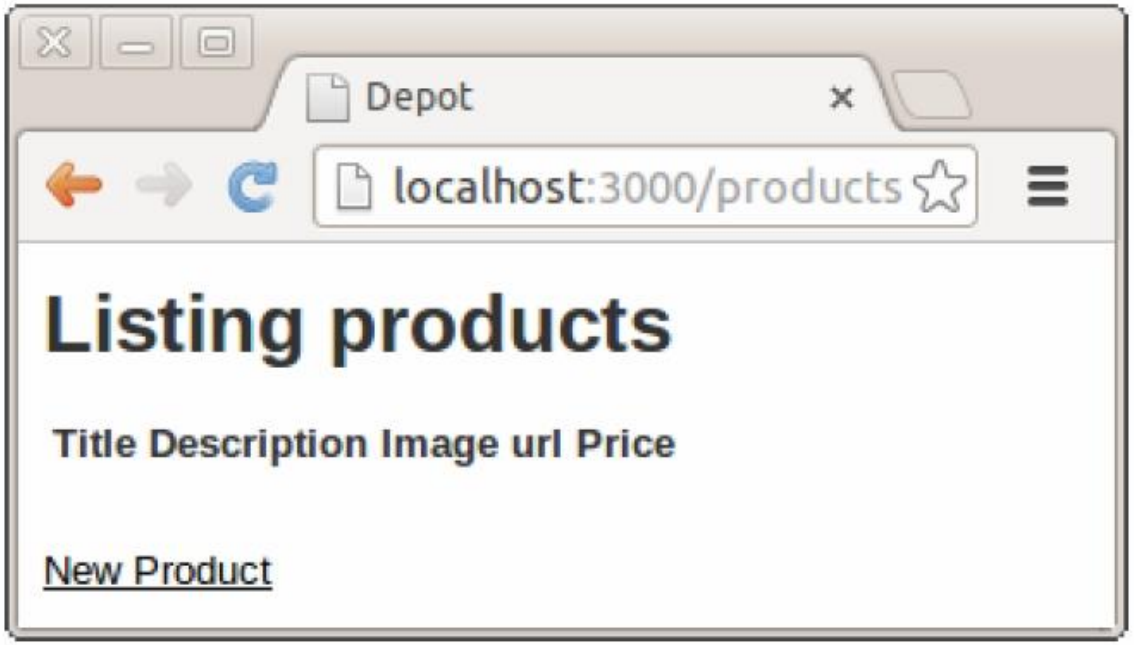
  - In our case, the *products* table is added to the database defined by *development* section of the *database.yml* file.
- All the groundwork has been done!

Seeing the list of products

- We are starting the server:

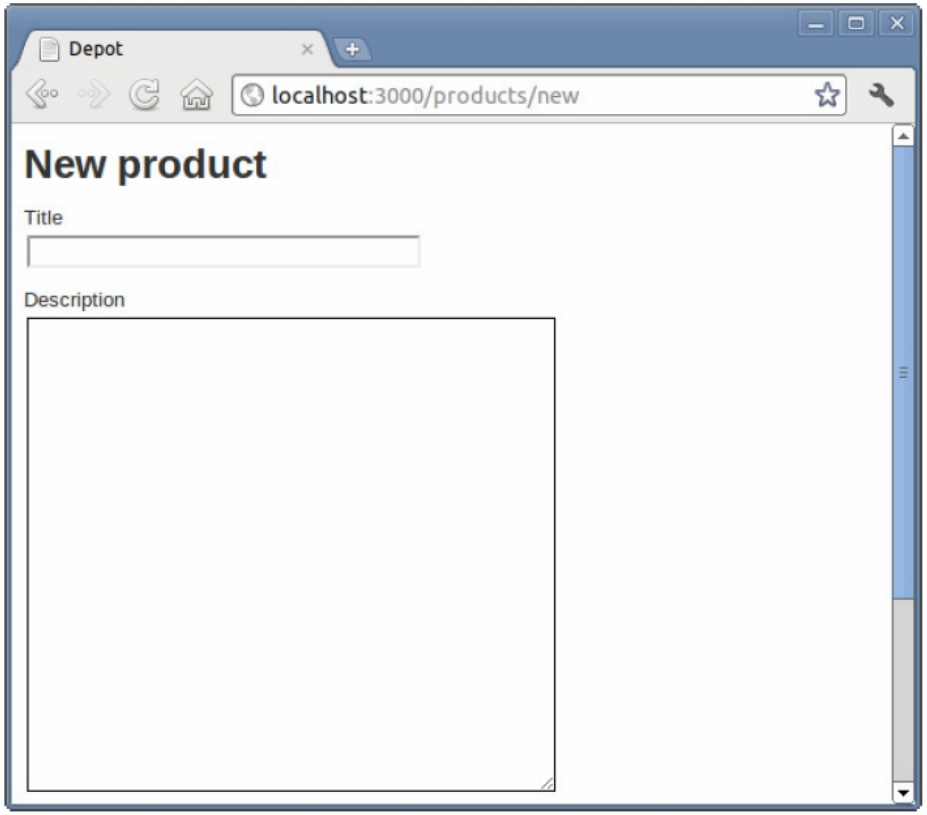
  depot> rails server
- Web server is on a local server on port 3000.
- To access the server, type in the browser:

  http://localhost:3000/products



Seeing the list of products

- After clicking the *New product* link:
  - These forms are simply HTML templates
  - We may change the number of lines in the description field (see the next slide)



Seeing the list of products

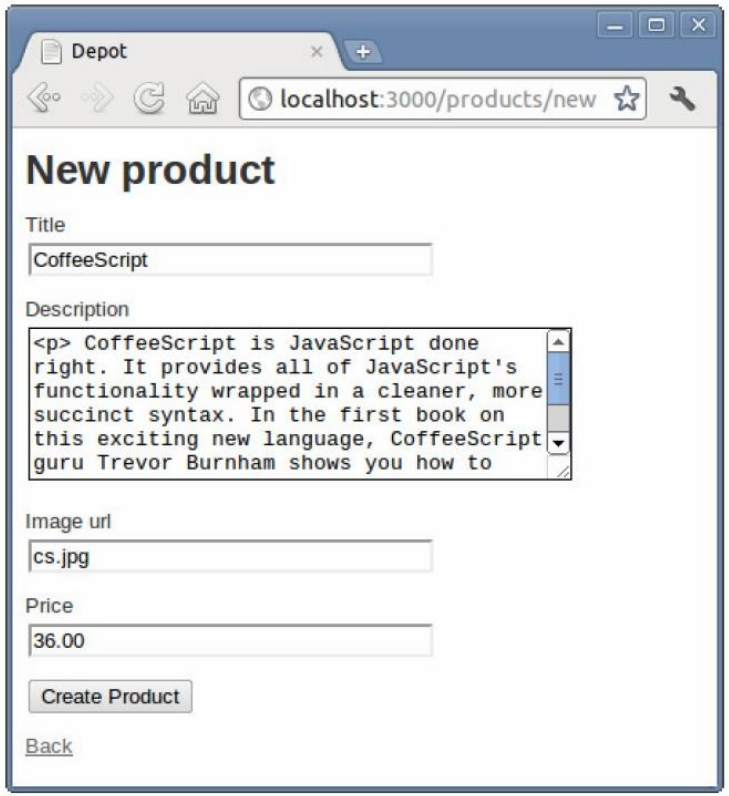
- File:

  depot_a/app/views/products/_form.html.erb

  - Line:

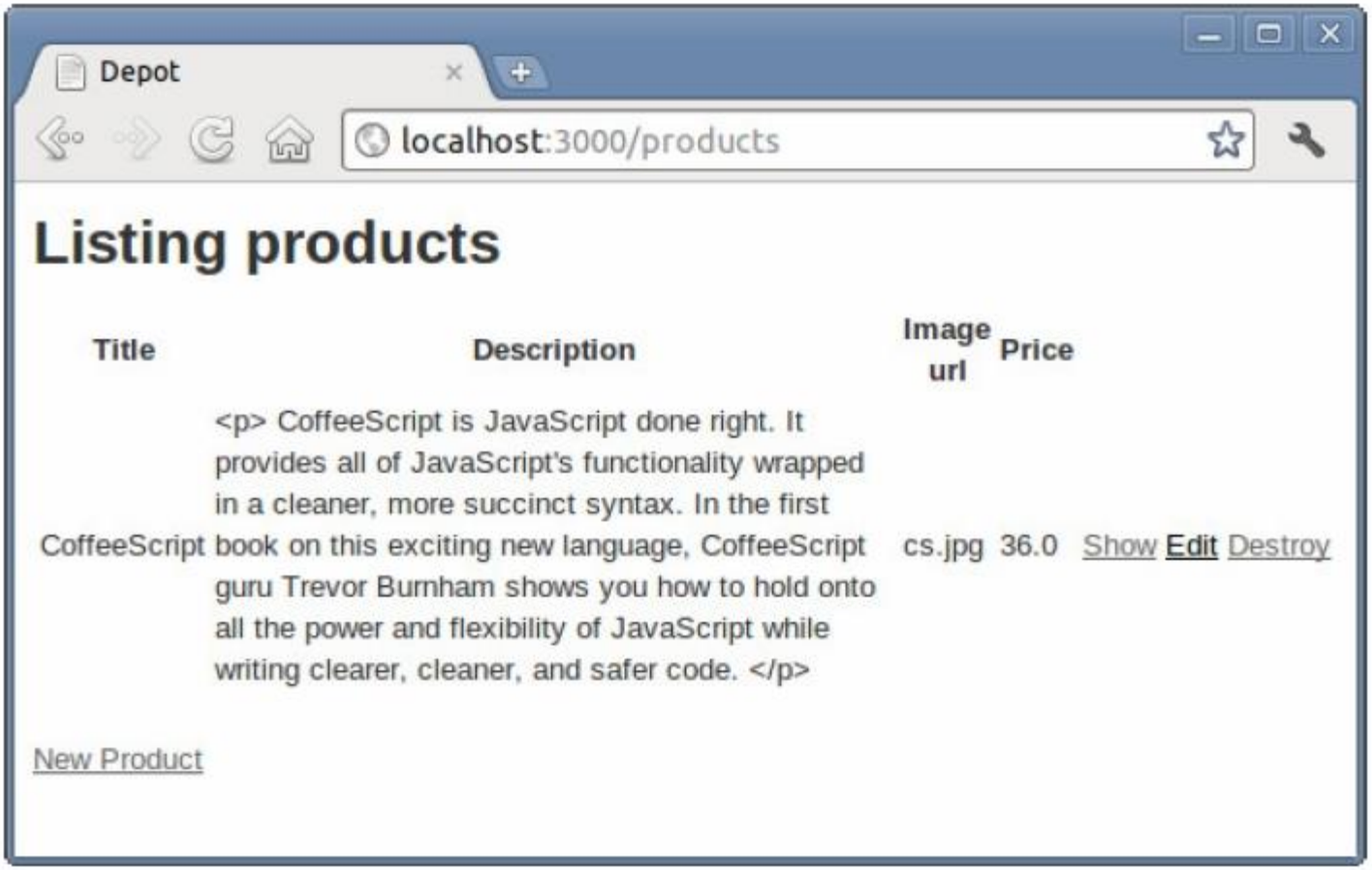
    <%= f.text_area :description, rows: 6%>

- Result is here:



Seeing the list of products

- Click the *Create* button and:



Seeing the list of products

- It is not the prettiest interface, but it works.
- We can show it to our client for approval!
- Testing:

  depot>rake test

  - In the output should be 2 lines that each say *0 failures, 0 errors*
  - This is for the unit, functional and integration tests that Rails generates.
  - You will run this command frequently to spot and track down errors.

Making the prettier listings

- The listings of all the products are ugly :(
- Can we also display the product image?
- To simplify your work, you may use the following test data and style sheets developed download them into corresponding places:
  - media.pragprog.com/titles/rails4/code/depot_b/db/seeds.rb
  - media.pragprog.com/titles/rails4/code/depot_b/public/images
  - media.pragprog.com/titles/rails4/code/depot_b/public/stylesheets/depot.css

Making the prettier listings

- We will modify the following file:

Download rails40/depot_a/db/seeds.rb

```
Product.delete_all
# . . .
Product.create!(title: 'Programming Ruby 1.9 & 2.0',
  description:
    %{<p>
        Ruby is the fastest growing and most exciting dynamic language
        out there. If you need to get working programs delivered fast,
        you should add Ruby to your toolbox.
      </p>},
  image_url: 'ruby.jpg',
  price: 49.95)
# . . .
```

- The full file is available from: http://media.pragprog.com/titles/rails4/code/rails40/depot_a/db/seeds.rb
- Images should be copied into the app/assets/images directory: http://media.pragprog.com/titles/rails4/code/rails40/depot_a/app/assets/images/

Making the prettier listings

- To populate your product with test data:

  ```
  depot> rake db:seed
  ```

- Now, it is time to link the CSS style sheets.

Making the prettier listings

- Next file to work with:

Download rails40/depot_a/app/assets/stylesheets/products.css.scss

```
// Place all the styles related to the Products controller here.
// They will automatically be included in application.css.
// You can use Sass (SCSS) here: http://sass-lang.com/

.products {
  table {
    border-collapse: collapse;
  }

  table tr td {
    padding: 5px;
    vertical-align: top;
  }

  .list_image {
    width:  60px;
    height: 70px;
  }
```

Making the prettier listings

- Next file to work with:

Download rails40/depot_a/app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
<head>
  <title>Depot</title>
  <%= stylesheet_link_tag    "application", media: "all",
    "data-turbolinks-track" => true %>
  <%= javascript_include_tag "application", "data-turbolinks-track" => true %>
  <%= csrf_meta_tags %>
</head>
<body class='<%= controller.controller_name %>'>
<%= yield %>
</body>
</html>
```

Making the prettier listings

- We will use a simple table-based template editing the file replacing the scaffold-generated view:

Download rails40/depot_a/app/views/products/index.html.erb

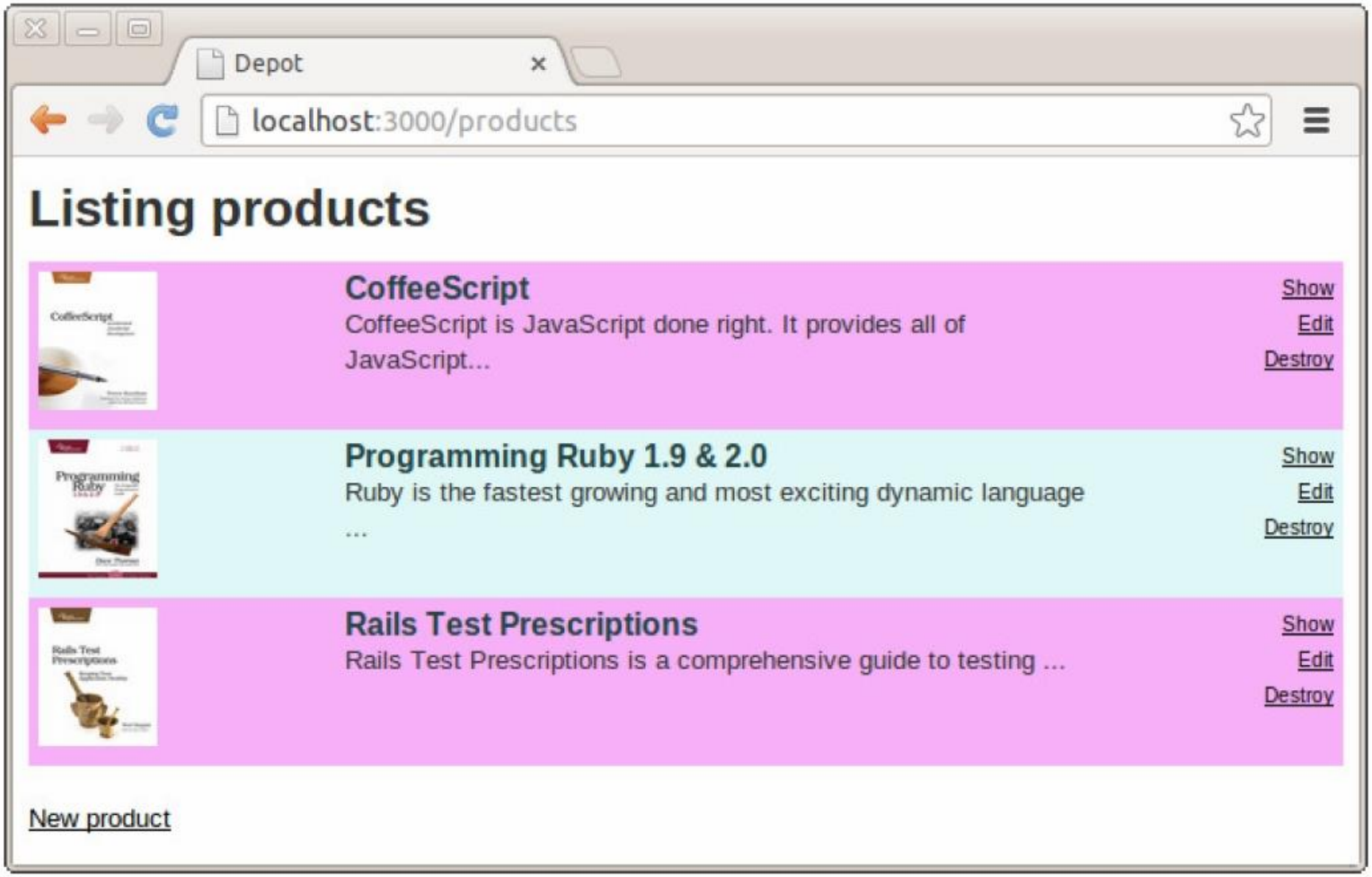
```
<h1>Listing products</h1>

<table>
<% @products.each do |product| %>
  <tr class="<%= cycle('list_line_odd', 'list_line_even') %>">
    <td>
      <%= image_tag(product.image_url, class: 'list_image') %>
    </td>
    <td class="list_description">
      <dl>
        <dt><%= product.title %></dt>
        <dd><%= truncate(strip_tags(product.description), length: 80) %></dd>
      </dl>
    </td>
    <td class="list_actions">
      <%= link_to 'Show', product %><br/>
      <%= link_to 'Edit', edit_product_path(product) %><br/>
      <%= link_to 'Destroy', product, method: :delete,
                  data: { confirm: 'Are you sure?' } %>
    </td>
  </tr>
<% end %>
</table>
<br />

<%= link_to 'New product', new_product_path %>
```

Making the prettier listings

- Now, results of our efforts:



Playtime

- To roll back the migration, use:

  depot> rake db:rollback

- Version controll:

  depot> git repo-config -- get-regexp user.*

  - This is to verify the configuration

- To initialize a repository, add all the files and commit them:

  depot> git init

  depot> git add .

  depot> commit –m "Depot Scaffold"

- To restore deleted or overwritten files:

  depot > git checkout .

Dot is part of the command

What we just did

- We created the development database
- We used migration to create and modify the schema in our development database
- We created *products* table and used the scaffold generator to write the application to maintain it.
- We updated an application wide layout as well as a controller-specific viewin order to show a list of products.