Ruby on Rails with Instant Rails

2010-01 SEG4110 Lab

University of Ottawa

November 21 2011

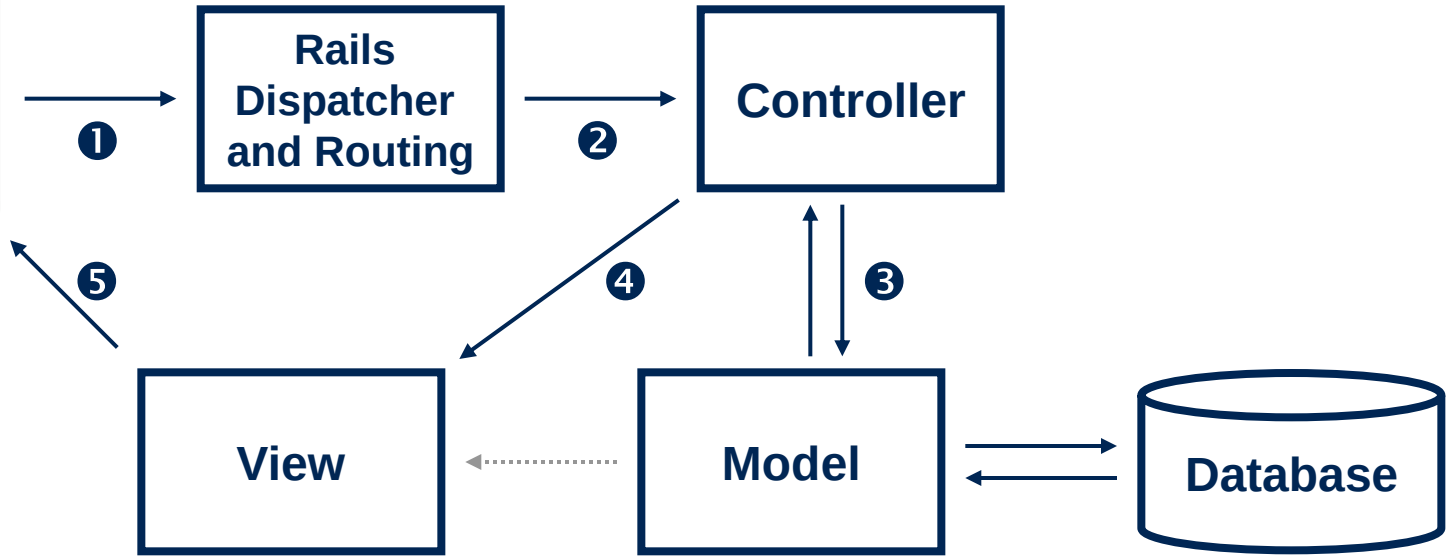
Table of Contents

Overview

- Development of database-driven web applications made easy
- Convention over Configuration
  - Almost no configuration files
  - Predefined directory structure
  - Naming conventions
- Best Practices
  - Model-View-Controller (MVC)
  - Don’t Repeat Yourself (DRY)
  - Continuous Testing

MVC Request Cycle

1. Request is made from browser (e.g. http://localhost:3000/stores/show/1)
2. Rails server running locally on port 3000 receives request and dispatcher is invoked. Rails tries to match a route (e.g. the default route ':controller/:action/:id' matches). Dispatcher instantiates stores controller and invokes its show method with the :id parameter set to 1.
3. Controller interacts with model to fetch store object with id=1 from db
4. Controller invokes view (e.g. the file show.html.erb)
5. View renders next browser screen



Instant Rails

- Ruby on Rails Distribution

- Instant Rails 3.0
  - Ruby 1.8.7
  - Rails 3.0
  - Web servers (WEBrick)
  - Databases (SQLite3)
  - Database Manager (phpMyAdmin)
  - Text Editor (SciTE)
- Instant Rails takes care of starting/stopping/configuring the web and database servers as well as the Rails applications

- Easiest installation for Windows
  - If you're working on Windows, you can quickly install Ruby and Rails with:

    http://railsinstaller.org/

Technologies and the MVC Request Cycle

Mongrel web server

routes (config\routes.rb)

scaffolding

Rails Dispatcher and Routing

Stores Controller :Application Controller

❶ ❷ ❸ ❹ ❺

config\database.yml

Editor:
SciTE
Testing:
Fixtures
ZenTest
(requires diff)

View

Store :ActiveRecord

stores table

*.html.erb
(embedded ruby)

partials

Migrations

MySQL

phpMyAdmin

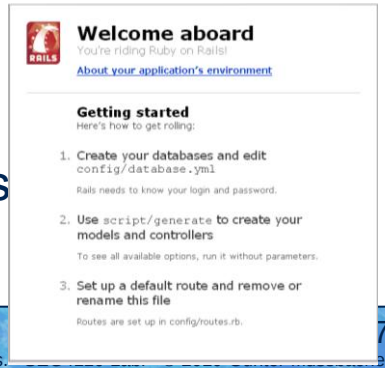
Validation

Associations

Eight Steps to Build a Simple Rails Application

Note: paths are relative to <RailsPath>\rails_apps\shoppinglist\

1. Generate skeleton of the Shopping List application
   - Open the command line in Windows
   - Type **rails new shoppinglist** (*shoppinglist* is the application name)
   - **cd shoppinglist**
2. Creating the Database
   - Type **rake db:create**
   - This will create your development and test SQLite3 databases inside the db/ folder.
3. Starting up the Web Server
   - In a new command line type **rails server**
   - This will fire up an instance of the WEBrick web server by default (Rails can also use several other web servers).
   - To see your application in action, open a browser window and navigate to **http://localhost:3000**. You should see Rails default information page:



Eight Steps to Build a Simple Rails Application

4. Generate application based on domain model
   - Go to the first console window
   - Type **rails generate scaffold Store name:string address:text** (*Store* is a model entity and *name:string address:text* are its attributes)
   - See folders app and db

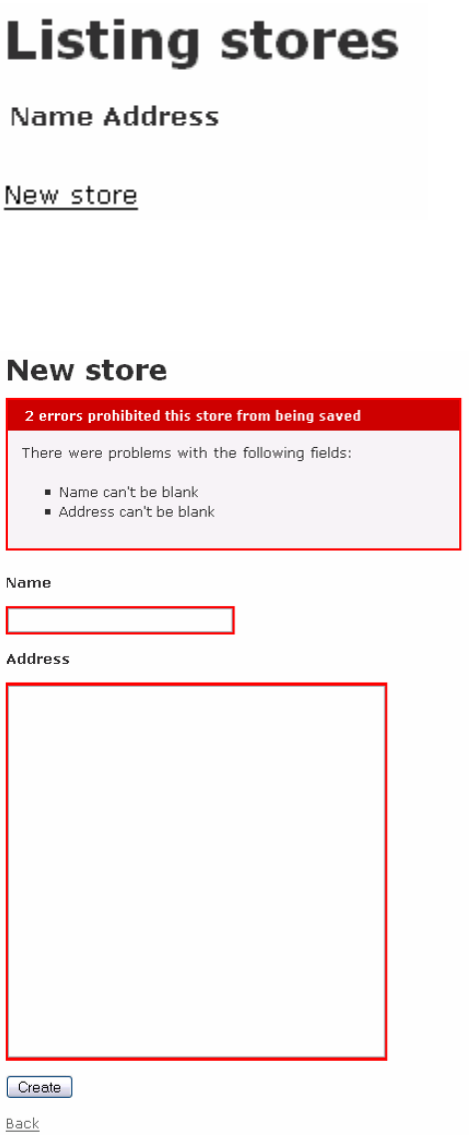
Eight Steps to Build a Simple Rails Application

5. Change default routing in config\routes.rb
   - Open file config\routes.rb
   - Uncomment root :to => ‘welcome#index'
   - Replace welcome with **home** and save the file
   - Rename the file for the default welcome screen from public\index.html to public\index.backup.html
   - Refresh the browser showing the welcome screen (you should get a Could not find table 'stores‘ error)
6. Synchronize database
   - Go to the first console window
   - Type **rake db:migrate**

Eight Steps to Build a Simple Rails Application

- Refresh the browser showing the error (you should see the stores interface as shown on the right)
- Try out the interface by creating new stores, editing stores, and destroying stores;

8. Input Validation
   - Open file app\models\store.rb
   - Insert a new line after the first line and add **validates_presence_of :name, :address** (*name, address* are the attributes to be validated)
   - Save the file
   - Go back to the browser and create a new store without entering a name and address and click Create (should see error as shown on the right)
   - Enter something in the input fields and click Create



Testing the Simple Rails Application

- Scaffolding also generates tests
- Fixtures (test\fixtures*.yml) describe the test data – Rails will set up the testing environment for each test based on the fixtures
- Unit / Functional / Integration Tests
  - We will be looking at functional tests

- In the first console window, **rake test** or just **rake** as test is the default

Submission Point 1

Archive your shopping list application (i.e., folder <RailsPath>\rails_apps\shoppinglist) in a zip file called LabRoR<YourFirstName><YourLastName>**1**.zip.

At the end of the lab, hand in this zip file as instructed on the lab handout.

Expand the Simple Rails Application: Associations

- Add ShoppingItem to the application

1. Generate application based on domain model
   - **rails generate scaffold ShoppingItem store_id:integer name:string**
2. Verify routing in config\routes.rb
   - Verify that **resources :shopping_items** is defined in the file
3. Synchronize database
   - **rake db:migrate**
4. Input Validation in file app\models\shopping_item.rb
   - Insert a new line after the first line and add:
     validates_presence_of :store_id, :name
   - Go back to the browser, type localhost:3000/shopping_items, and create, edit, and destroy shopping items
   - Destroy all shopping items before proceeding

Expand the Simple Rails Application: Associations

- Add an association between Store and ShoppingItem
  (this requires changes to the model, route, controller, and view)
- Each Shopping Item belongs to one Store
- One Store can have many Shoppping Items

1. Changes in Model
   - Insert a new line in file app\models\store.rb after the first line and add: **has_many :shopping_items, :dependent => :destroy**
   - Insert a new line in file app\models\shopping_item.rb after the first line and add: **belongs_to :store**
2. Changes in Routing: file config\routes.rb
   - Open up the config/routes.rb file again, you will see an entry that was added automatically for stores near the top by the scaffold generator, resources :stores, edit it as follows:

```
resources :stores do
   resources : shopping_items
end
```

Expand the Simple Rails Application: Associations

3. Generating a Controller:

   rails generate controller ShoppingItem

- app/controllers/shopping_item_controller.rb – The controller
- app/helpers/shopping_item_helper.rb – A view helper file
- test/functional/shopping_item_controller_test.rb – The functional tests for the controller
- test/unit/helpers/shopping_item_helper_test.rb – The unit tests for the helper
- app/views/shopping_items/ – Views of the controller are stored here
- app/assets/stylesheets/shopping_item.css.scss – Cascading style sheet for the controller
- app/assets/javascripts/shopping_item.js.coffee – CoffeeScript for the controller

Expand the Simple Rails Application: Associations

- Types of variables in Ruby
  - var is a local variable
  - Var is a constant (can be set once)
  - @var is an instance variable
  - $var is a global variable
  - @@var is a class variable (static variable)

- Arrays in Ruby
  - myarray = [‘first’, ‘second’, ‘third’…]
- Iterating over an array in Ruby
  - ```
    i = 0
    myarray.each do |cellvalue|
      puts “cell #{i}: #{cellvalue}”
      i = i + 1
    end
    ```


    ```

# Expand the Simple Rails Application: Associations

## 4. Changes in View

- Shopping items can be created after the creation of the Store.
- So first, we'll wire up the Store show template (/app/views/stores/show.html.erb) to let us create a new shopping item:

```
<h2>Add a Shopping Item:</h2>

<%= form_for([@store, @store.shopping_items.build]) do |f| %>

  <div class="field">

    <%= f.label :name %>

    <%= f.text_field :name %>

  </div>

  <div class="actions">

    <%= f.submit %>

  </div>

<% end %>
```

Expand the Simple Rails Application: Associations

5. Add the following code to the ShoppingItemsController. (Replace the create procedure with this code)

```
# POST /shopping_items
# POST /shopping_items.json

def create
  @store = Store.find(params[:store_id])
  @shopping_item =
 @store.shopping_items.create(params[:shopping_item])
  redirect_to store_path(@store)
 end
```

Expand the Simple Rails Application: Associations

6. Add the following code to (/app/views/stores/show.html.erb) to display the list of shopping items for a each store.

```
<h2>Shopping Items:</h2>
<% @store.shopping_items.each do |shopping_item| %>
  <p>
    <b>Name:</b>
    <%= shopping_item.name %>
  </p>
<% end %>
```


Overview of Associations

- One-to-zero-or-one
  - class Company
    has_one :board_of_directors
    end

    class BoardOfDirectors
    belongs_to :company
    end
- One-to-many
  - class Company
    has_many :employees
    end

    class Employee
    belongs_to :company
    end
- Many-to-many
  - class Project
    has_and_belongs_to_many
    :employees
    end

    class Employee
    has_and_belongs_to_many
    :projects
    end

Submission Point 2

Archive your shopping list application (i.e., folder <RailsPath>\rails_apps\shoppinglist) in a zip file called LabRoR<YourFirstName><YourLastName>**2**.zip.

At the end of the lab, hand in this zip file as instructed on the lab handout.

Expand the Simple Rails Application: Security

- If you were to publish your Shopping Application online, anybody would be able to add, edit and delete your stores.
- Rails provides a very simple HTTP authentication system that will work nicely in this situation.
- In the **StoresController** we need to have a way to block access to the various actions if the person is not authenticated, here we can use the Rails **http_basic_authenticate_with** method, allowing access to the requested action if that method allows it.
- To use the authentication system, we specify it at the top of our **StoresController**, in this case, we want the user to be authenticated on every action, except for index and show, so we write that:

```
http_basic_authenticate_with :name => “miguel", :password =>
  "secret", :except => :index
```


Submission Point 3

Archive your shopping list application (i.e., folder <RailsPath>\rails_apps\shoppinglist) in a zip file called LabRoR<YourFirstName><YourLastName>**3**.zip.

At the end of the lab, hand in this zip file as instructed on the lab handout.

