Case Study: Data Prototyping

CITS3403 Web & Internet Technologies

Reference: Ruby et al, Chapter 6

Slides © 2009 Cara MacNish and © 2011, 2012 Wei Liu, University of Western Australia.
Returning to the Depot case study...

- First development task - web interface to maintain products
  - create new products
  - edit existing products
  - delete unwanted products
- Develop in small iterations..
Iteration A1: Get Something Running

• Tasks
  – Create Rails app
  – Create database(s) to hold information
  – Point Rails to database(s)
  – Create table to hold product information

• Create Rails application

  > rails new depot
Iteration A1: Get Something Running

• Create the Database
  – SQLite3 is default database for versions 2.0.2 onwards
  – if using something else (eg mysql) read Ruby et al, Section 6.1
  – we will use SQLite3 → illustrates the benefits of “convention over configuration”!

• Typical approaches to creating databases...
  – use Data Definition Language (or SQL), eg create table etc
    • disadvantage: lots of low-level work, manual roll-back, etc
  – use interactive tools to create and maintain schemas, eg phpMyAdmin
    • disadvantage:
      – difficult to roll back your changes, essentially irreversible
      – difficult to deploy - ensure same changes to development and production databases
Database Migrations

- **migrations** - changes to database expressed in database-independent terms
  - update both scheme and data
  - stored in UTC time-stamped files
  - apply migrations, roll them back
  - provides a version-controlled history of database schema
  - can re-create it
- Rails
  - generator to create new model
    ‣ automatically creates a migration to create corresponding table!
Creating Model, Views and Controller

- Rails generates “scaffolding”
  - all the default directories, view templates, stylesheets, controllers, testers,...
  - database model and migration file

```
depot> rails generate scaffold product

> title:string description:text image_url:string
```

- “product” (singular) generates model `Product` and database table `products` (“convention over configuration”)
- remainder specifies attributes of a product (correspond to columns in the table)
Creating Model, Views and Controller

depot> rails generate scaffold product
> title:string description:text image_url:string
exists app/models/
exists app/controllers/
exists app/helpers/
create app/views/products
exists app/views/layouts/
exists test/functional/
exists test/unit/
exists public/stylesheets/
create app/views/products/index.html.erb
create app/views/products/show.html.erb
create app/views/products/new.html.erb
create app/views/products/edit.html.erb
create app/views/layouts/products.html.erb
create public/stylesheets/scaffold.css
create app/controllers/products_controller.rb
create test/functional/products_controller_test.rb
create app/helpers/products_helper.rb
route map.resources :products

continued...
dependency model
exists app/models/
exists test/unit/
exists test/fixtures/
create app/models/product.rb
create test/unit/product_test.rb
create test/fixtures/products.yml
create db/migrate
create db/migrate/2011006031921_create_products.rb

• 2011006031921_create_products.rb is the migration - just a Ruby file:

```ruby
class CreateProducts < ActiveRecord::Migration
  def self.up
    create_table :products do |t|
      t.string :title
      t.text :description
      t.string :image_url
      ...
```
Migrating the Database

• Use a utility, or “assistant”, called *rake*

```bash
depot> rake db:migrate
(in /home/cits3403/cits3403y/depot)
== CreateProducts: migrating
=======================================
-- create_table(:products)
  -> 0.0033s
== CreateProducts: migrated (0.0043s) ========================================
```

– looks for all migrations not applied to the database and applies them
– in this case creates the *products* table
Checking the Database

- We can check the database manually...

```bash
> sqlite3 db/development.sqlite3
SQLite version 3.6.20
Enter "help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .databases
seq  name            file
--- -------------- ------------- ----------------------------
0    main            /home/cits3403/cits3403y/depo/db/development.sqlite3

sqlite> .tables
products           schema_migrations

sqlite> select * from schema_migrations;
20111006031921

sqlite> select * from products;
```

```bash
sqlite>
```
Running the Application

- From just those 3 commands Rails has created a basic, working application..!
Running the Application

New product

Title
Clock Tower

Description
<p>Historic clock tower, one only. Don't miss this one!</p>

Image url
http://www.uwa.edu.au/__data/assets/

Create

Back
Running the Application

Listing products

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
<th>Image url</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Tower</td>
<td>&lt;p&gt;Historic clock tower, one only. A timely purchase. Don't miss this one!&lt;/p&gt;</td>
<td>/crawley_winthrop.jpg</td>
<td>Show Edit Destroy</td>
</tr>
<tr>
<td>Sunken Garden</td>
<td>&lt;p&gt;This one has sunk a little, but still in great shape. Ideal for weddings, parties, anything!&lt;/p&gt;</td>
<td>/images/sunken.jpg</td>
<td>Show Edit Destroy</td>
</tr>
</tbody>
</table>
Iteration A2: Adding a Missing Column

- Our client notices there are no prices for products
  - need to add a column to database table
    ‣ create a migration...
    ```
    depot> rails generate migration add_price_to_product price:decimal
      active_record
    create  db/migrate/20111006053934_add_price_to_product.rb
    ```
    ‣ edit the migration...
    ```
    class AddPriceToProduct < ActiveRecord::Migration
      def self.up
        add_column :products, :price, :decimal, :precision => 8, :scale => 2, :default => 0
      end

      def self.down
        remove_column :products, :price
      end
    end
    ```
Iteration A2: Adding a Missing Column

- run rake
  - Rails knows which version the database is up to, so applies any newer migrations
- edit views
  - index.html.erb
  - new.html.erb
  - edit.html.erb
  - show.html.erb
Iteration A2: index.html.erb (original)

```html
<h1>Listing products</h1>
<table>
  <tr>
    <th>Title</th>
    <th>Description</th>
    <th>Image url</th>
  </tr>
  <% @products.each do |product| %>
    <tr>
      <td><%= product.title %></td>
      <td><%= product.description %></td>
      <td><%= product.image_url %></td>
    </tr>
  <% end %>
</table>
<br/>
<%= link_to 'New Product', new_product_path %>
```
Iteration A2: Adding a Missing Column

- Eg. *index.html.erb*

```html
<h1>Listing products</h1>
<table>
  <tr>
    <th>Title</th>
    <th>Description</th>
    <th>Image url</th>
    <th>Price</th>
  </tr>
  <% @products.each do |product| %>
  <tr>
    <td><%=h product.title %></td>
    <td><%=h product.description %></td>
    <td><%=h product.image_url %></td>
    <td><%=h product.price %></td>
  </tr>
  <% end %>
</table>
```
Iteration A3: Validating Input

- Our client notices anything, or nothing, can be entered in the fields and committed to the database
  - products entered in database must have
    - non-empty title and description
    - valid URL for the image
    - valid price
- Where to put validation?
  - model layer is gatekeeper of database
  - all data goes via model whether it comes from user form or other code in our application
The Model Class

- `app/models/product.rb`

```ruby
class Product < ActiveRecord::Base
end
```

- all database mapping, creating, updating, searching, etc, inherited from parent class (`ActiveRecord::Base`) provided in Rails

- Use provided Rails methods (`validators`)
  
  ```ruby
  validates_presence_of :title, :description, :image_url
  ```

  - checks fields are present and non-empty
  
  ```ruby
  validates_numericality_of :price
  ```

  - checks price is a valid number

- Methods pop up an error if they don’t validate...
Validation

New product

1 error prohibited this product from being saved:
- Price is not a number

Title
Programming Ruby 1.9

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

Submit
Create Product
Back
Custom Validation Methods

• Write a method *(price_must_be_at_least_a_cent)* in *Product* model class

  - **protected**, since it shouldn’t be called from outside model

• Pass name of method to ActiveRecord::Base.validate method

  ```ruby
  validate :price_must_be_at_least_a_cent
  ```

  – Rails will call this method before saving instances of product

```ruby
protected
def price_must_be_at_least_a_cent
  errors.add(:price, 'should be at least 0.01')
  if price.nil? || price < 0.01
end
```
More Validation

validates_uniqueness_of :title

- checks no other row has same title

validates_format_of :image_url,

  :with => %r{\.\((gif|jpg|png)\)$}i,

  :message => 'must be a URL for GIF, JPG ' +

    'or PNG image.(gif|jpg|png)'

- matches image URL field with regular expression

  * %r{...} - regular expression, i - case insensitive
Iteration A4: Prettier Listings

- Add stylesheet `depot.css` to `depot/public/stylesheets`

  /* Global styles */
  #notice {
    border: 2px solid red;
    padding: 1em;
    margin-bottom: 2em;
    background-color: #f0f0f0;
    font: bold smaller sans-serif;
  }

  /* Styles for products/index */

  #product-list table {
    border-collapse: collapse;
  }

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

  ...
Linking the Stylesheet

- Notice that *.html.erb templates contain no XHTML <head> section
  - separate file used to create standard page environment for all "products" pages - products.html.erb
  ‣ Rails layout
    • lives in layouts directory
Linking the Stylesheet

- `depot/app/views/layouts/products.html.erb`

```erb
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="content-type" content="text/html;charset=UTF-8" />
<title>Products: <%= controller.action_name %></title>
<%= stylesheet_link_tag 'scaffold', 'depot' %>
<%= javascript_include_tag :defaults %>
</head>
<body>
<p style="color: green"><%= flash[:notice] %></p>
<%= yield %>
</body>
</html>
```
Replacing the Scaffold-Generated View

- `depot/app/views/products/index.html.erb...`
  ```html
  <div id="product-list">
  <h1>Listing products</h1>
  
  <table>
  <% for product in @products %>
  <tr class="<%= cycle('list-line-odd', 'list-line-even') %>">
  </tr>
  </table>
  
  <%= image_tag product.image_url, :class => 'list-image' %>
  
  <td class="list-description">
  <dl>
  <dt><%= product.title %></dt>
  <dd><%= truncate(product.description.gsub(/<.*?>/,''), 80) %></dd>
  </dl>
  
  </td>
  ```
Replacing the Scaffold-Generated View

- depot/app/views/products/index.html.erb

```erb
  <td class="list-actions">
    <%= link_to 'Show', product %><br/>
    <%= link_to 'Edit', edit_product_path(product) %><br/>
    <%= link_to 'Destroy', product,
      :confirm => 'Are you sure?',
      :method => :delete %>
  </td>
</tr>
<% end %>
</table>
</div>

<br />

<%= link_to 'New product', new_product_path %>
```
Replacing the Scaffold-Generated View

• Notice a number of built-in Rails features
  – alternating colours by setting CSS class to either `list-line-even` or `list-line-odd` using helper method `cycle`
  – `gsub` used to remove XHTML tags, and helper `truncate` to display first 80 characters
  – `h` to ensure remaining XHTML is “escaped”
  – `link_to ‘Destroy’` pops up dialogue box for confirmation
Replacing the Scaffold-Generated View
Review

• Database creation and access
  – database and model set up by scripts provided
• Migrations
  – more scripts, enable roll-back, recreation (version control)
• “Convention-over-configuration” at work
  – Rails creates scaffold of a working system
  – tailor as required