Rails: Routes

Lecture 27
Recall: Rails Architecture
Configuration

- Need to map an HTTP request (verb, URL, parameters) to an application action (a method in a Ruby class)
  - Framework invokes the method, passing in parameters from HTTP request as arguments
  - Results in an HTTP response, typically with an HTML payload, sent back to client's browser
- These mappings are called *routes*
- Defined in `config/routes.rb`
  - Ruby code, but highly stylized (another DSL)
  - Checked top to bottom for first match
Basic Route

- Pattern string + application code
  - In config/routes.rb
  - Pattern string usually contains segments
    ```ruby
    get 'status/go/:system/memory/:seg',
    to: 'reporter#show'
    ```
- Matches any HTTP request like
  ```ruby
  GET /status/go/lander/memory/0?page=3
  ```
- Result:
  - Instantiates `ReporterController`
  - Invokes `show` method
  - Provides an object called `params`
  ```ruby
  params = { :system => 'lander',
             :seg => '0',
             :page => '3' }
  ```
Default Values

- Special symbols for segments
  - :controller - the controller class to use
  - :action - the method to invoke

- Example route
  ```ruby
  get ':controller/go/:action/:system'
  ```

- Matches any HTTP request like
  ```ruby
  GET /reporter/go/show/lander?page=3
  ```

- Result:
  - Instantiates `ReporterController`
  - Invokes `show` method
  - Provides an object called `params`
  ```ruby
  params = { :system => 'lander',
             :page => '3',
             # also :controller & :action }
  ```
Customizing Routes

- Recognize different HTTP verb(s)
  - `get, put, post, delete`
  - Alternative: `match via: [:get, :post]`

- Optional segments with ()
  - `get ':controller(/:action(/:id))'`

- Default values
  - `get 'photos/:id', to: 'photos#show',
    defaults: { format: 'jpg' }`
REST

- REpresentational State Transfer
  - An architectural style for web applications
  - Maps database operations to HTTP requests

- Small set of database operations (CRUD)
  - Create, read, update, delete

- Small set of HTTP verbs, with fixed semantics (e.g., idempotence)
  - GET, POST, PUT, DELETE

- The protocol is stateless

- "Resource": bundle of (server-side) state
  - Each resource is identified by a URL
Resources

- A resource could be an individual *member*
  - Example: a single student
  - Corresponds to a row in a table

- A resource could be a *collection* of items
  - Example: a set of students
  - Corresponds to a table

- In REST, resources have URLs
  - Each member element has its own URL
    - http://quickrosters.com/students/42
  - A collection has its own URL
    - http://quickrosters.com/students
Read Collection: GET

GET /students HTTP/1.1
Host: quickrosters.com

Request
Read Collection: GET

GET /students HTTP/1.1
Host: quickrosters.com

Request
Read Collection: GET

Listing students

Fname Lname Buckid
Marco Pantani 22352022 Show Edit Destroy
Primo Carrera 334132 Show Edit Destroy
Cher 34822039 Show Edit Destroy

New Student
HTML Source (GET Collection)

<h1>Listing Students</h1>
<table>
  <tr>
    <th>Fname</th>
    <th>Lname</th>
    <th>Buckid</th>
    <th colspan="3"></th>
  </tr>
  <tr>
    <td>Primo</td>
    <td>Carnera</td>
    <td>334432</td>
    <td><a href="/students/3">Show</a></td>
    <td><a href="/students/3/edit">Edit</a></td>
    <td><a href="/students/3" data-confirm="Are you sure?" data-method="delete" rel="nofollow">Destroy</a></td>
  </tr>
  <a href="/students/new">New Student</a>
Read Member: GET

GET /students/3

Request
## Minimal Set of Routes (R)

<table>
<thead>
<tr>
<th>Collection</th>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/students</code></td>
<td><code>/students/42</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>List all members</td>
<td>Show info about a member</td>
</tr>
<tr>
<td>PUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Minimal Set of Routes (CR)

<table>
<thead>
<tr>
<th>Collection /students</th>
<th>Member /students/42</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>List all members</td>
</tr>
<tr>
<td></td>
<td>Show info about a member</td>
</tr>
<tr>
<td>PUT</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
</tr>
</tbody>
</table>

- How to map “create member” action?
  - Member doesn’t yet exist ➔ target is collection
  - Creation is not idempotent ➔ verb is...
## Minimal Set of Routes (CR)

<table>
<thead>
<tr>
<th></th>
<th>Collection /students</th>
<th>Member /students/42</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET</strong></td>
<td>List all members</td>
<td>Show info about a member</td>
</tr>
<tr>
<td><strong>PUT</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>POST</strong></td>
<td>Create a new member</td>
<td></td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- How to map “create member” action?
  - Member doesn’t exist ➔ target is collection
  - Creation is not idempotent ➔ verb is...
## Minimal Set of Routes (CRU)

<table>
<thead>
<tr>
<th>Collection</th>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>/students</td>
<td>/students/42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>List all members</td>
</tr>
<tr>
<td></td>
<td>Show info about a member</td>
</tr>
<tr>
<td>PUT</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>Create a new member</td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
</tr>
</tbody>
</table>

- How to map “update member” action?
  - Target is a member
  - Update overwrites, so it is idempotent...
## Minimal Set of Routes (CRU)

<table>
<thead>
<tr>
<th>Action</th>
<th>Collection /students</th>
<th>Member /students/42</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>List all members</td>
<td>Show info about a member</td>
</tr>
<tr>
<td>PUT</td>
<td></td>
<td>Update member</td>
</tr>
<tr>
<td>POST</td>
<td>Create a new member</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- How to map “update member” action?
  - Target is a member
  - Update overwrites, so it is idempotent...
## Minimal Set of Routes (CRUD)

<table>
<thead>
<tr>
<th>Collection</th>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>/students</td>
<td>/students/42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>List all members</td>
</tr>
<tr>
<td>PUT</td>
<td>Update member</td>
</tr>
<tr>
<td>POST</td>
<td>Create a new member</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete this member</td>
</tr>
</tbody>
</table>

- Delete action destroys a member
### Minimal Set of Routes

<table>
<thead>
<tr>
<th>Collection /students</th>
<th>Member /students/42</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Show info about a member</td>
</tr>
<tr>
<td>PUT</td>
<td>Update member</td>
</tr>
<tr>
<td>POST</td>
<td>Create a new member</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete this member</td>
</tr>
</tbody>
</table>

#### Implications
- You can't delete a collection
- No idempotent operations on collection
Typical Workflow: Delete

- How does one destroy a member?
  - Need to issue an HTTP request:
    - `DELETE /students/4`

- Protocol:
  - GET the collection to see the list
  - Click a button next to one item in the list to issue a DELETE for that member

- Alternative:
  - GET the member to see the details
  - Click a button to issue a DELETE for that member
GET List, DELETE Member

GET /students

Listing students

<table>
<thead>
<tr>
<th>Fname</th>
<th>Lname</th>
<th>Buckid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marco</td>
<td>Pantani</td>
<td>22352022</td>
</tr>
<tr>
<td>Primo</td>
<td>Camera</td>
<td>334432</td>
</tr>
<tr>
<td>Cher</td>
<td></td>
<td>34822039</td>
</tr>
</tbody>
</table>

New Student

DELETE /students/4
Typical Workflow: Create

- How does one issue a POST on collection?
  - GET a (blank) form
  - Fill in fields of form
  - Click a button to submit, resulting in the POST

- That first GET is a new route
  - GET on the collection
  - But instead of a list of members, the result is a form to be filled in and submitted
GET Blank Form, POST the Form

Listing students

Fname Lname Buckid
Marco Pantani 22352022 Show Edit Destroy
Primo Carrera 334432 Show Edit Destroy
Cher 34822039 Show Edit Destroy

New Student

GET "a blank form"

POST /students
lname: ...etc
# Standard Set of Routes

<table>
<thead>
<tr>
<th>Collection</th>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>/students</td>
<td>/students/42</td>
</tr>
<tr>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>1. List all members</td>
<td>1. Show info about a member</td>
</tr>
<tr>
<td>2. Form for entering a new member's data</td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td>Update member</td>
</tr>
<tr>
<td>POST</td>
<td>Create a new member</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete this member</td>
</tr>
</tbody>
</table>
<h1>Listing Students</h1>
<table>
  <tr>
    <th>Fname</th>
    <th>Lname</th>
    <th>Buckid</th>
    <th colspan="3"></th>
  </tr>
  ...
  <tr>
    <td>Primo</td>
    <td>Carnera</td>
    <td>334432</td>
    <td><a href="/students/3">Show</a></td>
    <td><a href="/students/3/edit">Edit</a></td>
    <td><a href="/students/3" data-confirm="Are you sure?" data-method="delete" rel="nofollow">Destroy</a></td>
  </tr>
  ...
</table>

<a href="/students/new">New Student</a>
Typical Workflow: Update

- How does one issue a PUT on a member?
  - GET a (populated) form
  - Edit the fields of the form
  - Click a button to send, resulting in the PUT

- That first GET is a new route
  - GET on a member
  - But instead of a display of information about that member, the result is a populated form to modify and submit
GET Filled Form, PUT the Form

Listing students

<table>
<thead>
<tr>
<th>Fname</th>
<th>Lname</th>
<th>Buckid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marco</td>
<td>Pantani</td>
<td>22352022</td>
</tr>
<tr>
<td>Primo</td>
<td>Carrera</td>
<td>334432</td>
</tr>
<tr>
<td>Cher</td>
<td></td>
<td>34822039</td>
</tr>
</tbody>
</table>

New Student

Editing student

Fname
Lname
Cher
Buckid
34822039

Update Student

Show | Back

GET "a populated form"

PUT /students/4
lname: ...etc
# Standard Set of Routes

<table>
<thead>
<tr>
<th></th>
<th><strong>Collection /students</strong></th>
<th><strong>Member /students/42</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET</strong></td>
<td>1. List all members</td>
<td>1. Show info about a member</td>
</tr>
<tr>
<td></td>
<td>2. Form for entering a new member's data</td>
<td>2. Form for editing an existing member's data</td>
</tr>
<tr>
<td><strong>PUT</strong></td>
<td></td>
<td>Update member</td>
</tr>
<tr>
<td><strong>POST</strong></td>
<td>Create a new member</td>
<td></td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td></td>
<td>Delete this member</td>
</tr>
</tbody>
</table>
HTML Source

...<h1>Listing Students</h1><table>
  <tr>
    <th>Fname</th>
    <th>Lname</th>
    <th>Buckid</th>
    <th colspan="3"></th>
  </tr>
  ...<tr><td>Primo</td><td>Carnera</td><td>334432</td><td><a href="/students/3">Show</a></td>
  <td><a href="/students/3/edit">Edit</a></td>
  <td><a href="/students/3" data-confirm="Are you sure?"
data-method="delete" rel="nofollow">Destroy</a></td>
  </tr>
  ...
</table><a href="/students/new">New Student</a>
Rails Resource-Based Routes

For a resource like :students, the action pack includes:

- 1 controller (StudentController)
- 7 routes (each with a method in controller)
- 4 Views (list of students, show 1 student, new, edit)

<table>
<thead>
<tr>
<th>HTTP Verb</th>
<th>URL</th>
<th>Resource</th>
<th>Method</th>
<th>Response (View)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/students</td>
<td>Collection</td>
<td>index</td>
<td>list all</td>
</tr>
<tr>
<td>POST</td>
<td>/students</td>
<td>Collection</td>
<td>create</td>
<td>show one</td>
</tr>
<tr>
<td>GET</td>
<td>/students/new</td>
<td>Collection</td>
<td>new</td>
<td>blank form</td>
</tr>
<tr>
<td>GET</td>
<td>/students/3</td>
<td>Member</td>
<td>show</td>
<td>show one</td>
</tr>
<tr>
<td>GET</td>
<td>/students/3/edit</td>
<td>Member</td>
<td>edit</td>
<td>filled form</td>
</tr>
<tr>
<td>PUT</td>
<td>/students/3</td>
<td>Member</td>
<td>update</td>
<td>show one</td>
</tr>
<tr>
<td>DELETE</td>
<td>/students/3</td>
<td>Member</td>
<td>destroy</td>
<td>list all</td>
</tr>
</tbody>
</table>
Defining Resource-Based Routes

- In RosterTool app's config/routes.rb

```ruby
Rails.application.routes.routes.draw do
  resources :students
  resources :faculty

end
```
Customizing Routes

- To change which 7 routes are created
  
  ```ruby
  resources :students, :except => [:update, :destroy]
  resources :grades, :only => [:index, :show]
  ```

- To specify a particular controller
  
  ```ruby
  resources :students, :controller => 'ugrads'
  ```

- To rename certain actions
  
  ```ruby
  resources :students, :path_names => {
    :create => 'enroll'
  }
  ```

- To add more routes to standard set
  
  - Add GET /students/:id/avatar (i.e. on member)
  - Add GET /students/search (i.e. on collection)
  
  ```ruby
  resources :students do
    get 'avatar', :on => :member
    get 'search', :on => :collection
  end
  ```
Segment Keys

- URL request has *arguments* for controller
  - Example: products/42
  - Pattern string: 'products/\:id'
- Segment key gets value when route matches
- Controller gets a hash (called *params*) of segment keys and their values
  - Example: *params[\:id]* is '42'
- Common case: Look up an item by id
  ```ruby
  def set_product
    @product = Product.find(params[\:id])
  end
  ```
Recognition vs Generation

- Dual problems
  - Recognize a URL (request for an action)
  - Generate a URL (a hyperlink or redirect)
- Routes used for both!
- Helper methods _path give relative paths
  - `photos_path` #=> /photos
  - `new_photo_path` #=> /photos/new
  - `edit_photo_path(:id)` #=> /photos/4/edit
  - `photo_path(:id)` #=> /photos/4
- And _url helpers give full URL
  - `photos_url` #=> http://faces.com/photos
Debugging

☐ To see the full list of routes

```
$ rake routes

Prefix  Verb URI                Controller#Action
photos  GET /photos            photos#index
        POST /photos            photos#create
photo   GET /photo/:id         photos#show
```

☐ To see helpers

```
$ rails console

> app.edit_photo_path 42
=> "/photos/42/edit"
```
Root Route

□ With no matching route, **GET** for **http://example.com** gets index.html from application's public directory

□ To customize landing page, 2 choices:
  - Remove (or replace) public/index.html
  - Add `root` route to config/routes.rb, pointing to a controller#action
    
    ```ruby
    root :to => "welcome#index"
    ```
Singleton Resources

- Declared with singular syntax
  ```resource :system```
- You get only 1 resource
  - Not two (collection and member)
  - Controller still plural (e.g., SystemsController)
- Only 6 standard routes
  - No index collection action to list members
  - `POST /system` -> `create`
  - `GET /system/new` -> `new`
  - `GET /system/edit` -> `edit`
  - `GET /system/` -> `show`
  - `PUT /system` -> `update`
  - `DELETE /system` -> `destroy`
Summary

- REST and CRUD
  - Create, read, update, destroy
  - Map data to resources
  - Map actions to HTTP requests (verb + URL)

- Routes
  - Connect HTTP request to specific method in a controller class
  - Defined in config/routes.rb
  - Resource based, or match-based
  - Dual problem: recognition and generation