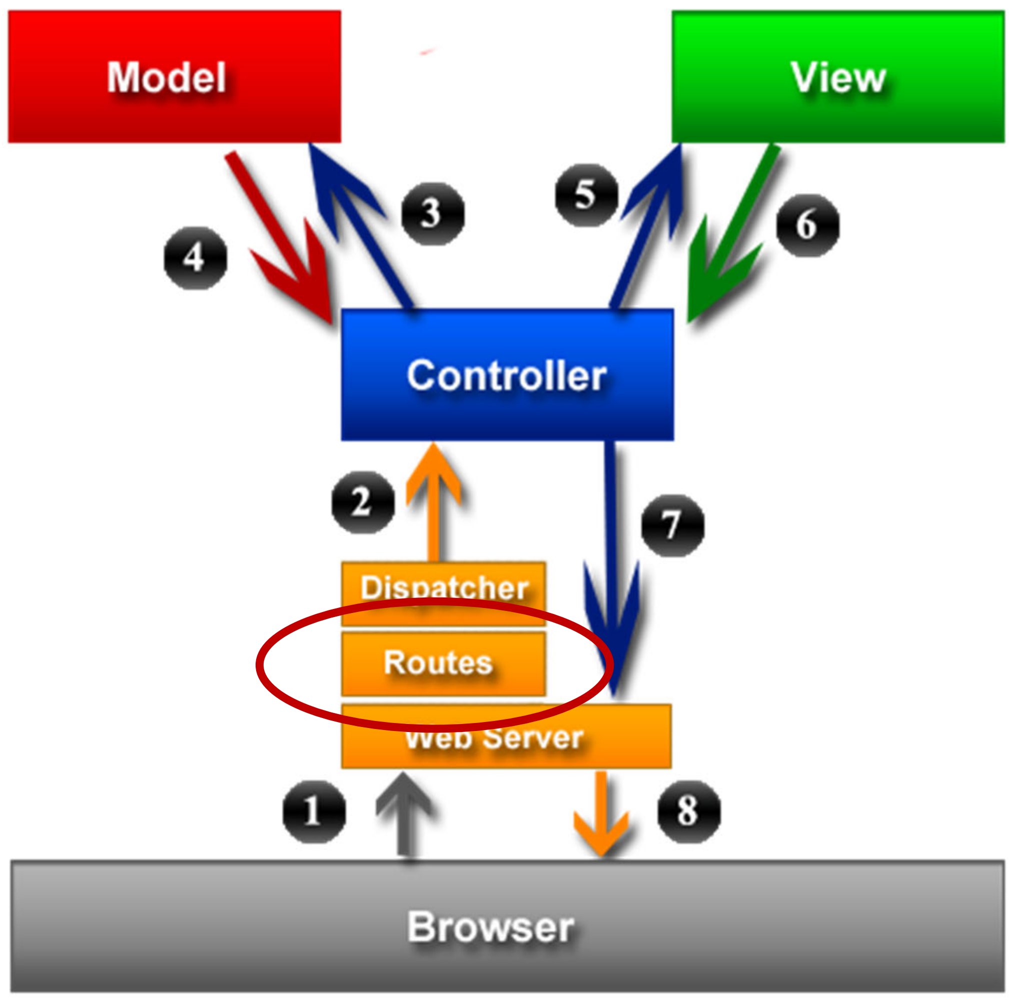
Rails: Routes

Lecture 30

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*
- Example route

```
get 'status/go/:system/memory/:seg',
    to: 'reporter#show'
```

- Matches any HTTP request like

```
GET /status/go/lander/memory/0?page=3
```

- Result:
  - Instantiates `ReporterController`
  - Invokes `show` method on that new instance
  - Provides an object called `params` (like a hash)

```
params = { system: 'lander',
           seg: '0',
           page: '3' }
```

Default Values

- Special segments
  - `:controller` - the controller class to use
  - `:action` - the method to invoke in that controller
- Example route

```
get ':controller/go/:action/:system'
```

- Matches *any* HTTP request like

```
GET /reporter/go/show/lander?page=3
```

- Result:
  - Instantiates `ReporterController`
  - Invokes `show` method on that new instance
  - Provides an object called `params`

```
params = { system: 'lander',
           page: '3',
           # also :controller and :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`
  - Each collection has its own URL

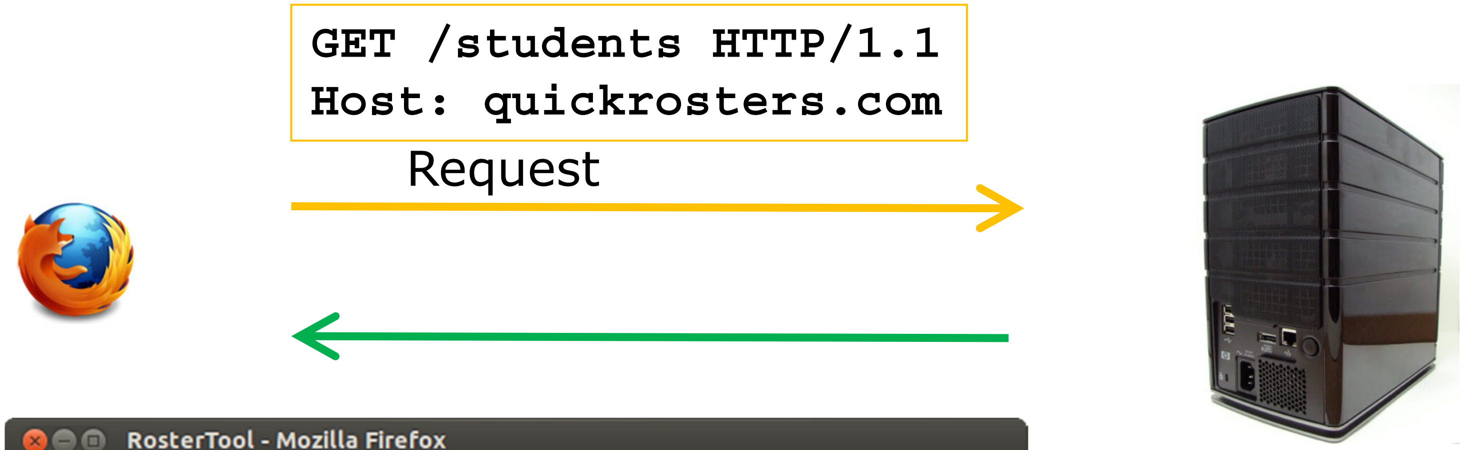
    `http://quickrosters.com/students`

Read Collection: GET

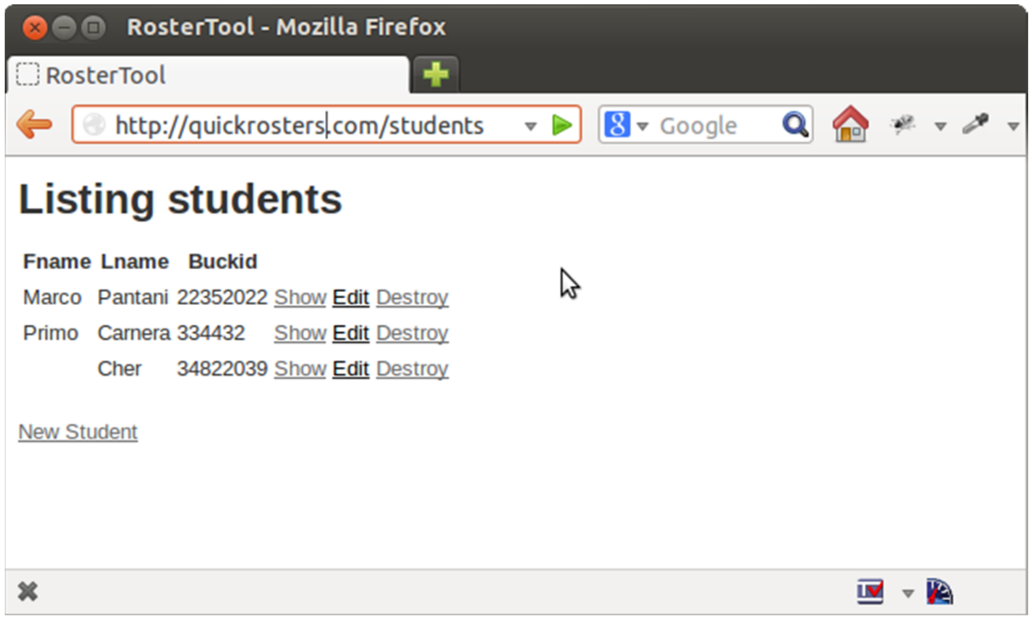
```
GET /students HTTP/1.1
Host: quickrosters.com
```

Request

Read Collection: GET



```
GET /students HTTP/1.1
Host: quickrosters.com
```



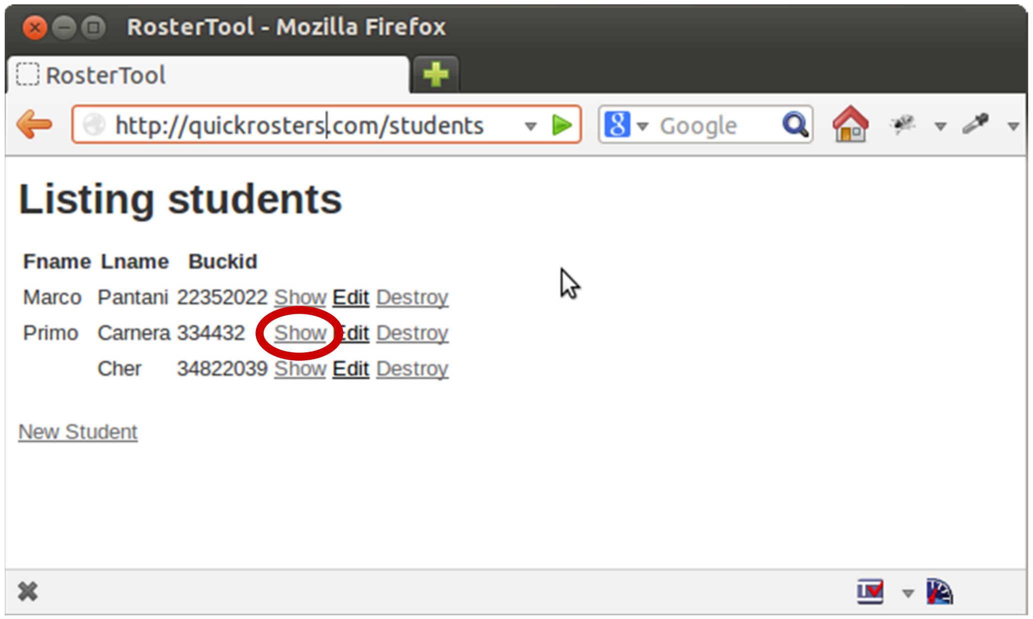
RosterTool - Mozilla Firefox
RosterTool
http://quickrosters.com/students
Google

Listing students

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

New Student

Read Collection: GET


RosterTool - Mozilla Firefox
RosterTool
http://quickrosters.com/students
Google

Listing students

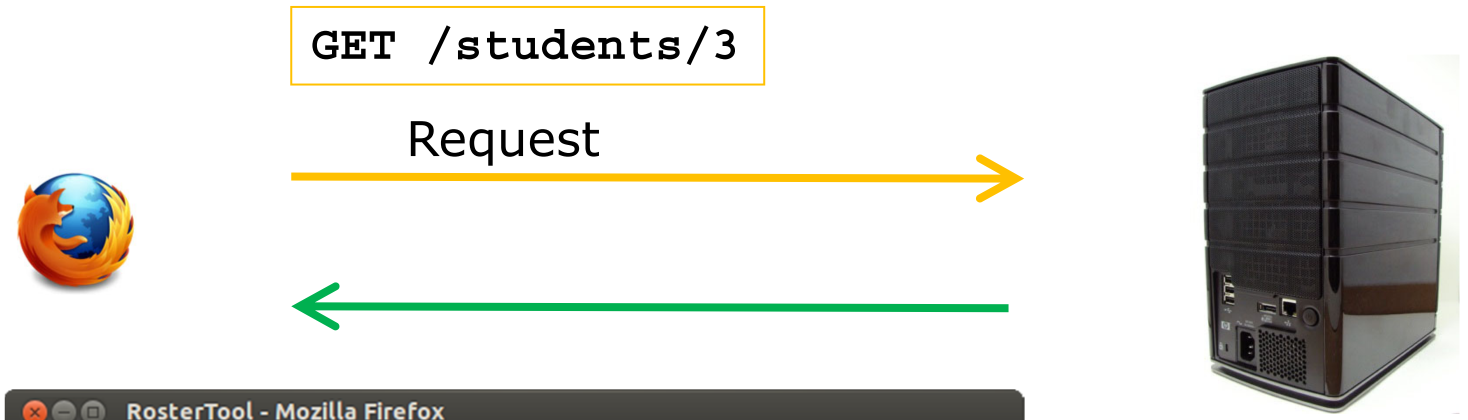
Fname	Lname	Buckid			
Marco	Pantani	22352022	Show	Edit	Destroy
Primo	Carnera	334432	Show	Edit	Destroy
	Cher	34822039	Show	Edit	Destroy

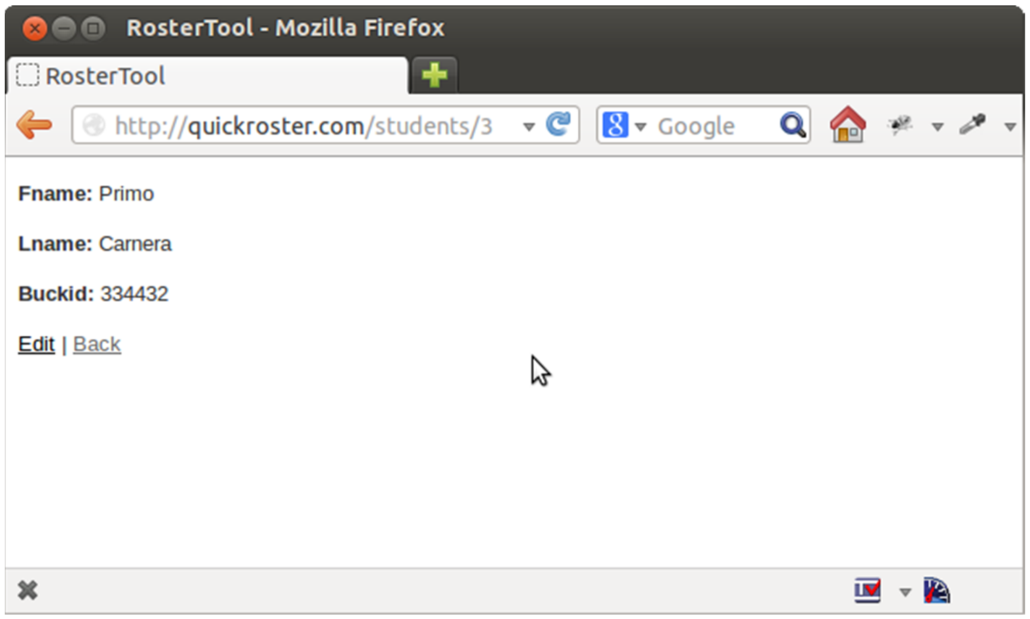
New Student

HTML Source (GET Collection)

```
…
<h1>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>
```

Read Member: GET




RosterTool - Mozilla Firefox
RosterTool
http://quickroster.com/students/3
Google

Fname: Primo

Lname: Carnera

Buckid: 334432

Edit | Back

Minimal Set of Routes (R)

	Collection `/students`	**Member** `/students/42`
GET	List all members	Show info about a member
PUT		
POST		
DELETE		

Minimal Set of Routes (CR)

	Collection `/students`	**Member** `/students/42`
GET	List all members	Show info about a member
PUT		
POST		
DELETE		

- How to map "create member" action?
  - Member doesn't exist ➔ target is collection
  - Creation is not idempotent ➔ verb is…

Minimal Set of Routes (CR)

	Collection `/students`	**Member** `/students/42`
GET	List all members	Show info about a member
PUT		
POST	Create a new member	
DELETE		

- How to map "create member" action?
  - Member doesn't exist ➔ target is collection
  - Creation is not idempotent ➔ verb is…

Minimal Set of Routes (CRU)

	Collection `/students`	**Member** `/students/42`
GET	List all members	Show info about a member
PUT		
POST	Create a new member	
DELETE		

- How to map "update member" action?
  - Target is a member
  - Update overwrites, so it is idempotent…

Minimal Set of Routes (CRU)

	Collection `/students`	**Member** `/students/42`
GET	List all members	Show info about a member
PUT		Update member
POST	Create a new member	
DELETE		

- How to map "update member" action?
  - Target is a member
  - Update overwrites, so it is idempotent…

Minimal Set of Routes (CRUD)

	Collection `/students`	**Member** `/students/42`
GET	List all members	Show info about a member
PUT		Update member
POST	Create a new member	
DELETE		Delete this member

- Delete action destroys a member

Minimal Set of Routes

	Collection `/students`	**Member** `/students/42`
GET	List all members	Show info about a member
PUT		Update member
POST	Create a new member	
DELETE		Delete this member

- Implications
  - You can't delete a collection
  - No idempotent operations on collection

Typical Workflow: Delete

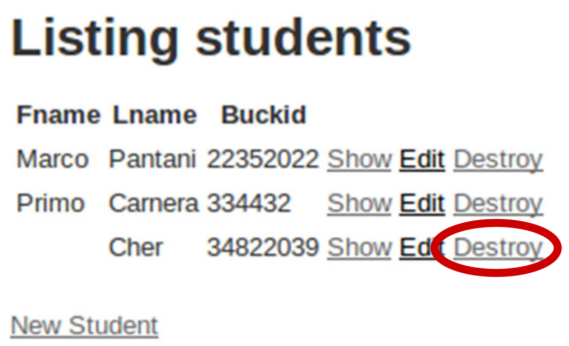
- How does one destroy a member?
  - Need to issue an HTTP request:

```
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

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

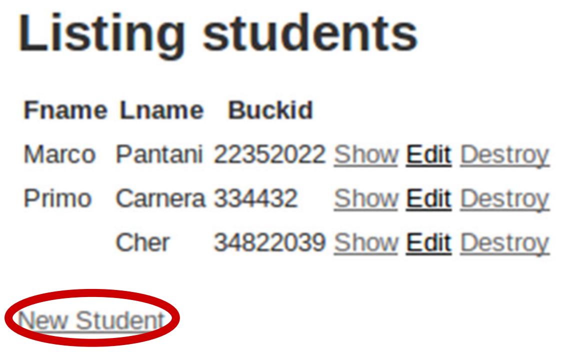
New Student

```
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

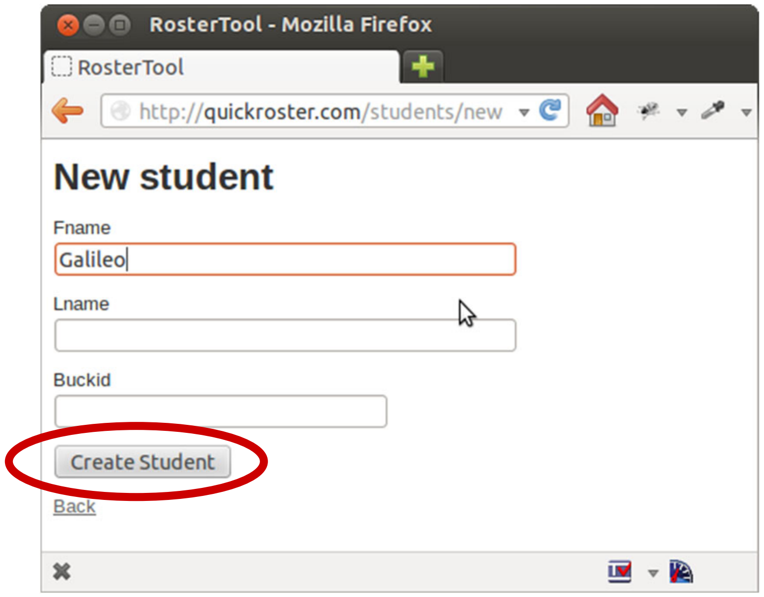


Listing students

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

New Student

`GET "a blank form"`



RosterTool - Mozilla Firefox

http://quickroster.com/students/new

New student

Fname: Galileo

Lname:

Buckid:

Create Student

Back

```
POST /students
lname: …etc
```

Standard Set of Routes

	Collection `/students`	**Member** `/students/42`
GET	1. List all members 2. Form for entering a new member's data	1. Show info about a member
PUT		Update member
POST	Create a new member	
DELETE		Delete this member

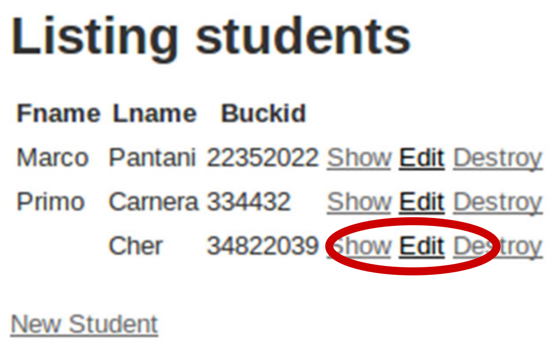
HTML Source

```
…
<h1>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

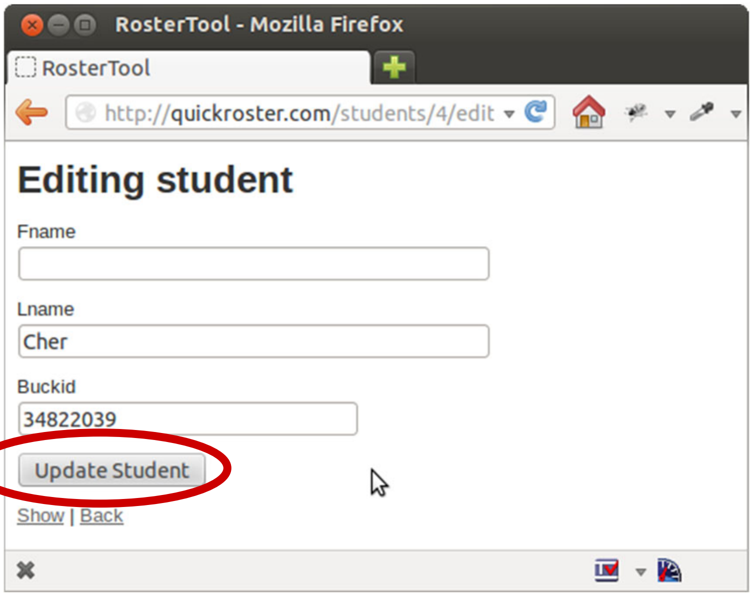


Listing students

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

New Student

GET *"a populated form"*



RosterTool - Mozilla Firefox
RosterTool
http://quickroster.com/students/4/edit

Editing student

Fname

Lname
Cher

Buckid
34822039

Update Student
Show | Back

```
PUT /students/4
lname: …etc
```

Standard Set of Routes

	Collection `/students`	**Member** `/students/42`
GET	1. List all members 2. Form for entering a new member's data	1. Show info about a member 2. Form for editing an existing member's data
PUT		Update member
POST	Create a new member	
DELETE		Delete this member

HTML Source

```
…
<h1>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 (StudentsController)
  - 7 routes (each with a method in controller)
  - 4 Views (list of students, show 1 student, new, edit)

HTTP Verb	URL	Resource	Method	Response (View)
`GET`	`/students`	Collection	`index`	list all
`POST`	`/students`	Collection	`create`	show one
`GET`	`/students/new`	Collection	`new`	blank form
`GET`	`/students/3`	Member	`show`	show one
`GET`	`/students/3/edit`	Member	`edit`	filled form
`PUT`	`/students/3`	Member	`update`	show one
`DELETE`	`/students/3`	Member	`destroy`	list all

Defining Resource-Based Routes

- In RosterTool app's `config/routes.rb`

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

Customizing Routes

- To change which 7 routes are created

```
resources :students, except:
                               [:update, :destroy]
resources :grades, only: [:index, :show]
```

- To specify a particular controller

```
resources :students, controller: 'ugrads'
```

- To rename certain actions

```
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)

```
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

```
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!
- For generation, route must be *named*

```
get 'status/:seg', to: 'reporter#show',
    as: :info
```

- Results in two helpers (_path, _url)

```
info_path(4)#=> "/status/4"
info_url(4) #=> "http://faces.com/status/4"
```

- Used with `link_to` to generate hyperlinks

```
link_to 'S', info_path(4), class: 'btn'
#=> "<a class='btn' href='/status/4'>S</a>"
```

Helper Methods for Resources

- Resource-based routes have names

```
photos_path         #=> /photos
photos_url          #=> http://faces.com/photos
new_photo_path      #=> /photos/new
photo_path(:id)     #=> /photos/4
edit_photo_path(:id) #=> /photos/4/edit
```

Name	HTTP	URL	Resource	Method
photos	GET	/photos	Collection	index
	POST	/photos	Collection	create
new_photo	GET	/photos/new	Collection	new
photo	GET	/photos/3	Member	show
edit_photo	GET	/photos/3/edit	Member	edit
	PUT	/photos/3	Member	update
	DELETE	/photos/3	Member	destroy

Debugging Routes and Helpers

- To see the full list of routes

```
$ rails routes
Prefix Verb URI             Contr#Action
  info GET  /status/:seg reporter#show
photos GET  /photos         photos#index
       POST /photos         photos#create
photo  GET  /photo/:id      photos#show
edit_photo GET /photos/:id/edit …
…etc…
```

- To see/use helpers in the console

```
$ rails console
> app.edit_photo_path(42)
=> "/photos/42/edit"
> helper.link_to "Click here",
    app.edit_photo_path(42)
=> "<a href="/photos/42/edit">Click here</a>
```

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:
  - Create public/index.html
  - Add **`root`** route to config/routes.rb, pointing to a controller#action (better)

```
root to: "welcome#index"
```

Singleton Resources

- Declared with singular syntax

  `resource :system`
- You get only 1 resource, not 2
  - Controller still plural (*e.g.*, SystemsController)
  - URLs are singular (e.g., /system/edit
- 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