In this lecture, we will discuss…

- ✧ Default Rendering
- ✧ Controller Based Rendering
- ✧ Partials

Representations - Rendering

- Default Rendering
  - Controller derives data to render
  - View defaults to action requested
  - Content form based on client specification

Representations - Rendering

- Controller Based Rendering
  - Controller derives data to render
  - Controller makes decision on action to render
  - Controller makes decision on the format to render
    - Supply default format or overriding client specification

Example

- Create an `Hello` example with one `hello` action/view

```
$ rails g controller Hello sayhello
$ find app | grep hello
app/views/hello
app/views/hello/sayhello.html.erb
app/controllers/hello_controller.rb
app/helpers/hello_helper.rb
app/assets/stylesheets/hello.scss
app/assets/javascripts/hello.coffee
```

Example - changes

- hello_controller.rb

```ruby
#app/controllers/hello_controller.rb
class HelloController < ApplicationController
  def sayhello
  end
end
```

- routes.rb

```ruby
#config/routes.rb
  get 'hello/sayhello'
```

- rake routes

```
$ rake routes | grep hello
  hello_sayhello GET    /hello/sayhello(.:format)                hello#sayhello
```

Default Rendering

- (action).(format).(handler suffix)
  - `sayhello.json.jbuilder` (JSON)
  - `sayhello.xml.builder` (XML)
  - `sayhello.text.raw` (RAW)

Default Rendering

- `#app/views/hello`
  - hello.json.builder → `json.msg @msg`
  - hello.xml.builder → `xml.msg @msg`
  - raw text message from: → `hello.text.raw`

Default Rendering - Demo

```
> response=MoviesWS.get("/hello/sayhello.json")
> response.header.content_type
 => "application/json"
> response.body
 => "{\"msg\":null}"
```

```
> response=MoviesWS.get("/hello/sayhello.xml")
> response.header.content_type
 => "application/xml"
> response.body
 => "<msg/>"
```

```
> response=MoviesWS.get("/hello/sayhello.txt")
> response.header.content_type
 => "text/plain"
> response.body
 => "raw text message from: app/views/hello/sayhello.text.raw\n"
```

Controller Derived Data

- Derive data from model
- Store in controller instance variable

```
#app/controllers/hello_controller.rb
class HelloController < ApplicationController
  def sayhello
    @msg="hello world"
  end
end
```

```
> MoviesWS.get("/hello/sayhello.json").response.body
=> "{\"msg\":\"hello world\"}"

> MoviesWS.get("/hello/sayhello.xml").response.body
=> "<msg>hello world</msg>\n"
```

Controller Action Rendering

- Controller can have more active say in action rendered
- Define a route and action that takes a parameter

```
#config/routes.rb
  get 'hello/say/:something' => "hello#say"
```

- Controller action method with decision logic

```
#/hello/say/:something
def say
  case params[:something]
  when "hello" then @msg="saying hello"; render action: :sayhello
  when "goodbye" then @msg="saying goodbye"; render action: :saygoodbye
  when "badword" then render nothing: true
  else
    render plain: "what do you want me to say?"
  end
end
```

Supported View Format

```
#app/views/hello/saygoodbye.json.jbuilder
json.msg1 @msg
json.msg2 "so long!"
```

```
#app/views/hello/saygoodbye.xml.builder
xml.msg do
  xml.msg1 @msg
  xml.msg2 "so long!"
end
```

Demo

- Request `something=hello`
  - supplied by caller
- Resulting View: sayhello
  - determined by the controller
- Format `json` format
  - Specified by caller

- Request `something=goodbye`
  - supplied by caller
- Resulting View: goodbye
  - determined by the controller
- Format `xml` format
  - Specified by caller

Partials - Example

- Partials allow you to easily organize and reuse your view code in a Rails application
- Filename starts with an underscore
- Action-independent
- Default path: `_(partial).(format).(handler suffix)`
  - `app/views/actors/_actor.json.jbuilder`
  - `app/views/actors/_actor.xml.builder`

Summary

- Different rendering approaches

What’s Next?

- Versioning

In this lecture, we will discuss…

- ✧ Versioning
- ✧ Vendor Media Type
- ✧ Backward compatible

Versioning

- Versioning – common practice in the industry
- `name` → `first_name` and `last_name`
- Backward compatible

```
class Actor
  field :first_name, type: String
  field :last_name, type: String

  #backwards-compatible reader
  def name
    "#{first_name} #{last_name}"
  end
```

Usage

- Legacy Users can still use the service

```
> pp MoviesWS.get("/actors/100.json").parsed_response
{"id"=>"100",
 "name"=>"sylvester stallone",
 "created_at"=>nil,
 "updated_at"=>"2016-01-05T22:19:42.642Z"}
```

- Problem – Not able to get the new additions via JSON (without breaking the legacy users)

Solution - Versioning

- Vendor Media Type
  - Define a different media type format
- Register the format (Ex: `v2json`)

```
#config/initializers/mime_types.rb`
# Be sure to restart your server when you modify this file.
# Add new mime types for use in respond_to blocks:
Mime::Type.register "application/vnd.myorgmovies.v2+json", :v2json
```

Demo – V2

- Access to v2 of the model

```
> response= MoviesWS.get("/actors/100.v2json")
> response.content_type
 => "application/vnd.myorgmovies.v2+json"

> pp JSON.parse(response.body)
{"id"=>"100",
 "first_name"=>"sylvester",
 "last_name"=>"stallone",
 "created_at"=>nil,
 "updated_at"=>"2016-01-05T22:19:42.642Z"}
```

Demo – V1

- Either define `v1json` from the start
- If not, default `json` as the “current” version

```
> response= MoviesWS.get("/actors/100.json")
> response.content_type
 => "application/json"

> pp JSON.parse(response.body)
{"id"=>"100",
 "name"=>"sylvester stallone",
 "created_at"=>nil,
 "updated_at"=>"2016-01-05T22:19:42.642Z"}
```

Summary

- Things are bound to change, consider versioning from the start

What's Next?

- Content Negotiation

In this lecture, we will discuss…

- ✧ Content Negotiation
- ✧ Accept and Accept-Encoding Headers

Content Negotiation

- Leave content type out of the URI
- Use HTTP Headers ("Accept" and "Content-Type") to express formats and encodings
  - `Content-Type:` what we are sending in
  - `Accept:` what we are willing to accept

Content Negotiation - Example

```
> response=MoviesWS.post("/directors",
    :body=>{:director=>{:id=>"300",:first_name=>"Tim",:last_name=>"Burton"}}.to_json,
    :headers=>{"Content-Type"=>"application/json","Accept"=>"application/json"})
> response.response
 => #<Net::HTTPCreated 201 Created  readbody=true>

> response.header["location"]
 => "http://localhost:3000/directors/300"
> response.header.content_type
 => "application/json"
```

Content Negotiation - Example

```
> pp JSON.parse(response.body)
{"id"=>"300",
 "first_name"=>"Tim",
 "last_name"=>"Burton",
 "created_at"=>"2016-01-06T01:16:26.302Z",
 "updated_at"=>"2016-01-06T01:16:26.302Z"}
```

Headers

- Accept: end-format we are willing to accept
- Accept-Encoding: intermediate form encoded on wire
- config/application.rb
  - `config.middleware.use Rack::Deflater`

Headers - Demo

- API Command

```
> response=MoviesWS.get("/directors",
    :headers=>{"Accept"=>"application/json","Accept-Encoding"=>"gzip"})
```

- HTTP Request

```
<- "GET /directors HTTP/1.1\r\n
Accept: application/json\r\n
Accept-Encoding: gzip\r\n
...
```

Summary

- Content Negotiation – server-based content negotiation to serve up the appropriate content

What's Next?

- Headers and Status codes

In this lecture, we will discuss…

- ✧ Headers
- ✧ Status codes

Headers

- Headers – used for concurrency checks or idempotence
- Cache Management
  - process if things are out of date
- Concurrency Management
  - process if things are up to date

State

- **`Etag`** (Entity Tags)
  - mechanism used to determine whether the entity or component (images, scripts, stylesheets, page content etc) in the browser's cache matches the one on the origin server
- **`Last-Modified`** TimeStamp
  - Indicates the most recent modification date/time

Conditions

- Conditions
  - `If-Match`: Etag
  - `If-None-Match`: Etag
  - `If-UnModified-Since`: Last-Modified Timestamp
  - `If-Modified-Since`: Last-Modified Timestamp

Headers – `fresh_when`

```
#app/controllers/movies_controller.rb
  def create
    @movie = Movie.new(movie_params)

    respond_to do |format|
      if @movie.save
        fresh_when(@movie)
        format.json { render :show, status: :created, location: @movie }
        format.v2json { render :show, status: :created, location: @movie }
      ...
```

Headers

DEMO

Status

- Etag and Last-Modified Headers are returned for `GET`

```
> response=MoviesWS.get("/movies/12347",:headers=>{"Accept"=>"application/vnd.myorgmovies.v2+json"})
> response.header["last-modified"]
=> "Wed, 06 Jan 2016 02:45:37 GMT"
> response.header["etag"]
=> "\"1db42608e19f6e50209190f8ac7470d2\""
```

- Etag and Last-Modified Headers are returned for `HEAD`

```
> response=MoviesWS.head("/movies/12347",:headers=>{"Accept"=>"application/json"})
> response.header["etag"]
=> "\"0258dce911d803a7ca3c394d83f52f9b\""
> response.header["last-modified"]
=> "Wed, 06 Jan 2016 02:45:37 GMT"
```

Updates

- Changes via `PUT`

```
> response=MoviesWS.put("/movies/12347",
    :body=>{:movie=>{:title=>"rocky2700"}}.to_json,
    :headers=>{"Content-Type"=>"application/json","Accept"=>"application/json"})
> response.response
 => #<Net::HTTPOK 200 OK  readbody=true>
```

- `Etag` and `Last-Modified` have changed

```
> response.header["etag"]
 => "\"fd9319ddeceb95b39af69b816705ee75\""
> response.header["last-modified"]
 => "Wed, 06 Jan 2016 03:22:33 GMT"
```

Concurrent Update Issue

- Nested Resource Update
  - Update Movie
  - Add Role to Movie
  - Add same Role to Movie (second time)
  - `Etag` is updated but `Last-Modified` will not
  - Can be fixed by using `touch`

IF_UNMODIFIED_SINCE

- If `HTTP_IF_UNMODIFIED_SINCE` is supplied
  - Use `fresh_when` to populate response with current `Etag` and `Last-Modified`
  - Continue if request date later than or equal to current state
  - Else report conflict and make no changes

Summary

- Cache Management and Concurrency Management can be successfully managed with status and headers

What’s Next?

- Caching