In this lecture, we will discuss…

- Foundation blocks for integrating Mongo with Ruby
  Driver Example

Setup and DAO

- ✧ Setup
  - New rails application, `mongoid`
- ✧ DAO class infrastructure
  - Connection, database, collection

CRUD Operations

- CRUD
  - `all`
  - `find`
  - `insert`
  - `update`
  - `delete`

Scaffolding

- Scaffolding
  - Controller
  - Views
  - *Note: Does not create the model class*

Pagination

- Paging – MVC application with paging
- `will_paginate`
- Will add page properties

Summary

- ✧ Build a working MVC example
- ✧ Simulate a middleware ORM consistent with the Rails ActiveModel framework

What's Next?

- ✧ Rails setup

In this lecture, we will discuss…

- Rails setup
- `mongoid` installation

New Application

- Create new application
  - `rails new zips`
- Add `mongoid` gem to Gemfile
  - Note: Mongoid will be covered in depth in Module 3
  - `gem 'mongoid', '~> 5.0.0'`

Configuring mongoid.yml

- Mongo database configuration
  - `rails g mongoid:config`
- mongoid.yml – database connection information
- `config/application.rb` - bootstraps mongoid within applications -- like rails console

Importing Data

- Import zips.json
  - `mongoimport --db zips_development --collection zips --file zips.json`

Start Rails Server

- Start Server
  - `rails s`
- *Note: Need to run the command inside the zips folder*

Rails Console

- Rails console
  - `rails c`
- Rails console – handy for poking around, basic calls, debug etc.

Summary

- Setting up rails application
- `mongoid` gem
- `rails` console

What's Next?

- DAO class infrastructure

In this lecture, we will discuss…

- ✧ DAO class infrastructure

DAO Class

- Model class
  - connects to MongoDB
  - Access to the collection (Example: “zips”)
- Consistent with ORM operations
  - find, insert, update, delete methods in the DAO class

DAO Class – zip.rb

```ruby
class Zip
  # convenience method for access to client in console
  def self.mongo_client
   Mongoid::Clients.default
  end

  # convenience method for access to zips collection
  def self.collection
   self.mongo_client['zips']
  end
end
```

Summary

- Simulating a middleware ORM that is consistent with the Rails ActiveModel framework
- “Data Access Object” and “Entity” at the same time

What’s Next?

- CRUD operations

In this lecture, we will discuss…

- CRUD

DAO Class – ORM Mapping

- `all` - maps to `find`
- `find` – maps to `find(hash)`
- `save` - maps to `insert_one`
- `update` – maps to `update_one`
- `destroy` – maps to `delete_one`

Adding Methods to Zips…..

- `all`
  - Return all documents in zips collection
- `self.all(prototype={}, sort={:population=>1}, offset=0, limit=100)`
  - Paging and Sorting

`find` and `save`

- `find id`
  - Return a specific instance for a given `id`
- `save`
  - Save the state of the current instance

`Update` and `destroy`

- ✧ `Update(updates)`
  - accepts as hash and performs an update on those values after accounting for any name mappings
- ✧ `destroy`
  - delete the document from the database that is associated with the instance's `:id`

Summary

- Basic CRUD operations

What's Next?

- Scaffolding

In this lecture, we will discuss…

- Model mixin
- `Scaffold` command
- Helpers

ActiveModel::Model Mixin Behavior

```
class Zip
  include ActiveModel::Model
...
  def persisted?
    !@id.nil?
  end
  def created_at
    nil
  end
  def updated_at
    nil
  end
```

- Check to see if a primary key has been assigned
- JSON marshalling will also expect a created_at and updated_at by default

scaffold command

```
$ rails g scaffold_controller Zip id city state population:integer
      create  app/controllers/zips_controller.rb
      invoke  erb
      create    app/views/zips
      create    app/views/zips/index.html.erb
      create    app/views/zips/edit.html.erb
      create    app/views/zips/show.html.erb
      create    app/views/zips/new.html.erb
      create    app/views/zips/_form.html.erb
      invoke  test_unit
      create    test/controllers/zips_controller_test.rb
      invoke  helper
      create    app/helpers/zips_helper.rb
      invoke    test_unit
      invoke  jbuilder
      create    app/views/zips/index.json.jbuilder
      create    app/views/zips/show.json.jbuilder
```

Helpers

```ruby
module ZipsHelper
  def toZip(value)
    #change value to a Zip if not already a Zip
    return value.is_a?(Zip) ? value : Zip.new(value)
  end
end
```

- `app/helpers/zips_helper.rb` - method will convert a Mongo document to a Ruby class instance

```
<% @zips.each do |zip| %>
  <% zip=toZip(zip) %>
```

`app/views/zips/index.html.erb`

Summary

- ✧ Scaffold command

What's Next?

- ✧ MVC Demo and Pagination

In this lecture, we will discuss…

- `show`
- `new` and `create`
- `edit` and `update`
- `destroy`
- `paging`

Show

```
#GET /zips/{id}
#GET /zips/{id}.json
  before_action :set_zip, only: [:show, :edit, :update, :destroy]
  def set_zip
    @zip = Zip.find(params[:id])
  end

  def show
  end
```

New and Create

```
#POST /zips/new
  def new
    @zip = Zip.new
  end

#POST /zips
  def create
    @zip = Zip.new(zip_params)

    respond_to do |format|
      if @zip.save
        format.html { redirect_to @zip, notice: 'Zip was successfully created.' }
        format.json { render :show, status: :created, location: @zip }
      else
        format.html { render :new }
        format.json { render json: @zip.errors, status: :unprocessable_entity }
      end
    end
  end
```

"New" returns an initial prototype to the form to start editing

"Create" accepts the results and creates new instance in the database

Edit and Update

```
http://localhost:3000/zips/00002/edit

#GET /zips/{id}
  before_action :set_zip, only: [:show, :edit, :update, :destroy]
  def set_zip
    @zip = Zip.find(params[:id])
  end

  def edit
  end

#PUT /zips/{id}
  def update
    respond_to do |format|
      if @zip.update(zip_params)
        format.html { redirect_to @zip, notice: 'Zip was successfully updated.' }
        format.json { render :show, status: :ok, location: @zip }
      else
        format.html { render :edit }
        format.json { render json: @zip.errors, status: :unprocessable_entity }
      end
    end
  end
```

"Edit" retrieved the instance from the database

"Update" found the instance in the database and applied the changes

Destroy

```ruby
#DELETE /zips/{id}
  def destroy
    @zip.destroy
    respond_to do |format|
      format.html { redirect_to zips_url, notice: 'Zip was successfully destroyed.' }
      format.json { head :no_content }
    end
  end
```

Paging

- `gem 'will_paginate', '~>`

```
<table>
...
  <tbody>
    <% @zips.each do |zip| %>
      <% zip=toZip(zip) %>
      <tr>
        <td><%= zip.id %></td>
...
      </tr>
    <% end %>
  </tbody>
</table>
<%= will_paginate @zips %>
```

"will_paginate" – adds page properties from the database

Paging (controller and model)

```ruby
def index
  #@zips = Zip.all
  @zips = Zip.paginate(:page => params[:page])
end
```

Controller passes the value to model

```ruby
def self.paginate(params)
  Rails.logger.debug("paginate(#{params})")
  page=(params[:page] ||= 1).to_i
  limit=(params[:per_page] ||= 30).to_i
  offset=(page-1)*limit

  #get the associated page of Zips -- eagerly convert doc to Zip
  zips=[]
  all({}, {}, offset, limit).each do |doc|
    zips << Zip.new(doc)
  end

  #get a count of all documents in the collection
  total=all({}, {}, 0, 1).count

  WillPaginate::Collection.create(page, limit, total) do |pager|
    pager.replace(zips)
  end
end
```

Will translate the will_paginate input to all() query inputs

Will translate document array results to will_paginate result

Summary

- MVC – proven model

What's Next?

- Module 2