Lesson 12: Handling file uploads & automating form building

Objectives

– Understand Ruby file input and output
– Implement file upload
– Paperclip
– Introduce FormBuilder
– Implement custom form builder
– Modify an existing form helper
– Form gems

How do you read and write from a file?

Open a file to append new files written to it, then close

```
# get file and set to append mode
file = File.new(“myfile.txt”, ”a”)

# write to file
file.write(”#{Time.now}\n”)

# close file
file.close
```

How do you read and write from a file?

Open a file to read and print lines from it, then close

```
# read the file
file = File.new(“myfile.txt”, "r")

# for each line do something
while(line = file.gets)
      puts line
end

# close the file
file.close
```

How can you temporarily store data to the file system?

Ruby Tempfile class creates a file in user’s /temp directory and opens it for write access

```
require ‘tempfile’
myfile = Tempfile.new(‘demo.txt’)
myfile.write(‘Some sample text’)
myfile.close
```

How do you enable a Rails form to upload a file?

Firstly, configure the form to encode multipart/form-data

```
<%= form_for(@review, html: { multipart: true }) do |f| %>
```

Implement a file_field element on the form

```
<div class="field">
    <%= f.label :image %><br />
    <%= f.file_field :image %>
</div>
```

How do you store uploaded files?

Code to handle upload commonly placed in the model (or a module to be included in the model)

Why not the controller?

Model code must:
- Determine where to store the data, and by what name
- Create the file, write data to it, and save it
- Save related data about the file, like its file extension
- Expose helper methods to provide the file path, URI, and whether the file exists

Where do you store uploaded files?

Commonly files are stored on the file system.

If storing images for web page access, store in the /public directory in a relevant sub-directory:

```
IMAGE_STORE = "#{Rails.root}/public/image_store"
```

Where do you store uploaded files?

May modify database to store additional file data, such as its type (file extension, .jpg, .png, etc.)

```
rails generate migration add_image_type_to_review

class AddImageTypeToReview < ActiveRecord::Migration
    def change
        add_column :reviews, :image_type, :string
    end
end
```

How do you store the uploaded file data?

Uploaded file is available in an automatic variable called file_data. If uploading through a file_field named :image, would assign file_data to that name in the model

```
def image=(file_data)
  unless file_data.blank?
    # assign file_data to instance variable
    # for use in store_image
    @file_data = file_data

    # assign image type to self.image_type
    self.image_type =
      file_data.original_filename.split('.').last.downcase
  end
end
```

How do you store the uploaded file data?

Create a directory and file, write the file data to it, and save

```ruby
def store_image
  if @file_data
    # create directory at IMAGE_STORE
    FileUtils.mkdir_p IMAGE_STORE

    # save image data to this location
    File.open(image_filename, 'wb') do |f|
      f.write(@file_data.read)
    end

    # nil file_data in memory so it won't be resaved
    @file_data = nil
  end
end
```

How do you store the uploaded file data?

Use a callback, like after_save, to call a method to store the uploaded data after other data has been saved

```
after_save :store_image
```

How do you store the uploaded file data?

Writer helper methods to provide the filename, URI, and whether a model has a related file

```
def image_filename
  “#{IMAGE_STORE}/#{id}.#{image_type}”
end

def image_uri
  “/image_store/#{id}.#{image_type}”
end

def has_image?
  File.exists? image_filename
end
```

How do you store the uploaded file data?

If a model has an image, display it in the show.html.erb view

```
<p>
  <b>Image:</b>
  <% if @review.has_image? %>
    <%= image_tag @review.image_uri %>
  <% else %>
    No image available
  <% end %>
</p>
```

Exercise:
implementing file upload and display

Paperclip

Let’s make it easier…

https://github.com/thoughtbot/paperclip

Installing Paperclip

On Mac:

Go to and follow the one-line instruction
http://mxcl.github.com/homebrew/

Then type in:

```
brew install imagemagick
```

On Windows:

Install the main version here:

http://www.imagemagick.org/script/binary-releases.php#windows

Installing Paperclip

Add it to your Gemfile:

```
gem ‘paperclip’, ‘~> 3.0’
```

Then either use the migration method:

```
add_attachment :users, :avatar
```

Or use a generator:

```
rails generate paperclip user avatar
```

Form building

What is a FormBuilder?

ActionView::Helpers::FormBuilder is the class which defines the various form helper methods. A new class can be derived from it, to add new helpers or override built in helpers with new behavior

```
class CustomFormBuilder < ActionView::Helpers::FormBuilder
  # stuff here
end
```

What is a FormBuilder?

The customized FormBuilder can be assigned to a form_for element to modify its helpers

```
<%= form_for(@review, html: { multipart: true },
    builder: CustomFormBuilder) do |f| %>
```

How could you create a new form helper?

In a class derived from FormBuilder declare a method which returns HTML, possibly using an existing helper. For example, create a select control pre-populated with a list of countries.

```
def country_select(method, options={}, html_options={})
   countries = []
   countries << ["Australia","Australia"]
   countries << ["Canada","Canada"]
   countries << ["UK","United Kingdom"]

   select(method, countries, options, html_options)
end
```

How do you use a new form helper?

Use the custom methods within that form_for element, along with the built in methods it inherits

```
<div class="field">
  <%= f.label :country %><br/>
  <%= f.country_select :country %>
</div>
```

Exercise:
Customizing FormBuilder to create a new form element

How can you change the behavior of existing form helpers?

In a custom FormBuilder, override an existing helper method by creating a new method with the same name.

Call the existing helper using the super keyword, but modify its results as appropriate.

How can you change the behavior of existing form helpers?

For example, modify the label helper to display a red asterisk if its field will be validated as required by the model

```
def label(method, options={}, html_options={})

  html_to_add = ""

  if options[:required]
      html_to_add << "<span class='required'>*</span>"
  end

  super(method, options) + html_to_add.html_safe

end
```

Exercise:
Extending an existing
form helper method using CSS

Form gems

Simple Form:
https://github.com/plataformatec/simple_form

Formtastic:
https://github.com/justinfrench/formtastic