CHAPTER 5

# Rails Controllers

## What is an actioncontroller?

Once the router has determined which controller it needs to deal with your request that controller is then responsible for dealing with that particular request, this may involve receiving data from the request, for example a new record need to be added to the database so the parameters – i.e. the fields and their associated data values, can then be passed to a model element to be dealt with.

This will typically also require a 'view' to be called to inform the user whether the action was successful or not.

A controller can be thought of as the 'middle man' between models and views.

It makes the model data available to the view so it can display that data to the user, and it saves or updates data from the user to the model.

It has no concept of the actual action itself but 'knows' how to call it and what to pass it in terms of data or some form of request.

Again, it also 'knows' what to do with the responses that it receives from one of the other elements.

The diagram of the Rails MVC in section 1 depicts this very well, some Rails programmer consider the controller to be the heart of the application.

An example of a controller class file **app/controllers/employee\_controller.rb** is shown below:

class EmployeeController < ApplicationController

def index

@name = params[:name]

unless @name.blank?

@employee = Employee.create({:name => @name})

end

@employees = Employee.all

end

end

There are many implicit links used by the controller that are 'invisble' to the programmer. This is the whole rationale behind MVC.

In the above example the controller obtains some data from the request by looking at the params object, in this case specifically the field identified as name.

It then calls the Model and 'asks' it to add an entry to the table again passing the data item onward.

It then makes another call to the model asking it to return all the entries in the table and stores them in an object for use elsewhere.

There is also an implicit link to a view called index.html.erb that the controller calls to provide the visual clues to the users.

The controller file can be from the very simple to the extremely complex depending on exactly the requirement of the application.

Fortunately, one of the feature or Rails is to be able to generate these controller files for you given some parameters to work with, we will see some example of this auto generation later in the course.

## Data Handling

Rails has added some data handling feature to the controller to support the concepts of 'DRY' and white listing.

These are implemented by two private 'call backs' now added to the end of a scaffold generated controller.

The code below shows the callbacks created.

private

# Use callbacks to share common setup or constraints between actions.

def **set\_employee**

@employee = Employee.find(params[:id])

end

# Never trust parameters from the scary internet, only allow the white list through.

def **employee\_params**

params.require(:employee).permit(:name, :phone, :department\_id)

end

The before\_action callback is invoked after the router has determine which method to call from the controller but before the action is invoked. It is mainly used to support 'DRY' by calling a private method that invokes the asscoiated model and retrieves the user row(s) from the table based on an id paramete as shown below.

before\_action :**set\_employee**, only: [:show, :edit, :update, :destroy]

The only: parameter say only for these actions listed.

The second callback is used to provide 'white listing'for the model, only the fields in the list are permitted to be accessed and the require object make them mandatory.

The example code below shows how the callback is invoked, any violations will cause an exception to be raised and the changes will not made to the data.

def create

@employee = Employee.new(**employee\_params**)

respond\_to do |format|

if @employee.save

format.html { redirect\_to @employee, notice: 'Employee was successfully created.' }

format.json { render :show, status: :created, location: @employee }

else

format.html { render :new }

format.json { render json: @employee.errors, status: :unprocessable\_entity }

end

end

end