Ruby on Rails

Ruby on Rails is a Web application framework for Ruby. It was first released to the public in July 2004.

Within months, it was a widely used development environment. Many multinational corporations are using it to create Web applications.

It is *the* standard Web-development framework for Ruby.

Model/View/Controller

All Rails applications are implemented using the Model/View/Controller (MVC) architecture.

Models are objects that represent the components of an application that perform information processing in the problem domain.

Models should represent "real world" entities:

- physical entities like a valve in a control system, or
- conceptual entities like a department in an office, or a contract between two businesses.

Views are objects that display some aspect of the model. They are the output mechanism for the models.

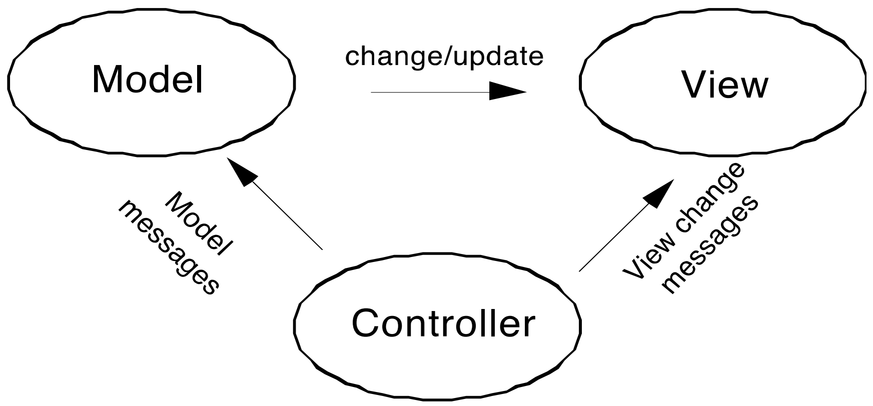
You could have a view that represents:

- the position of the valve or the temperature of a chemical vat (graphical)
- cells in a spreadsheet (tabular)
- the terms and conditions of a contract (textual)
- product installation instructions (video or audio)

Controllers are objects that control how user actions are interpreted. They are the input mechanism for the views and models.

For example, the interpretation of a double-click on a temperature gauge would be handled by the controller notifying the model in a way it agrees to respond to.

MVCs come in a triad, with communication between the components occurring as follows:



Whenever the state of a model changes, the view needs to display itself differently.

For instance, consider a View that represents a car's engine temperature.

If the engine temperature goes up (say, because a fan belt breaks) the gauge showing the temperature will need to redisplay its new value in response to that change.

In Rails, when a project is created, it is given folders for model, view, and controller classes.

A Rails application accepts an incoming request from a Web page, then gives it to a *router*. The router parses the URL and eventually directs the request to a particular controller.

For example, the URL might be something like **`http://expertiza.ncsu.edu/users/show/108`**.

This means to show the various fields in the entry for User 108 (name, e-mail address, role). In this case,

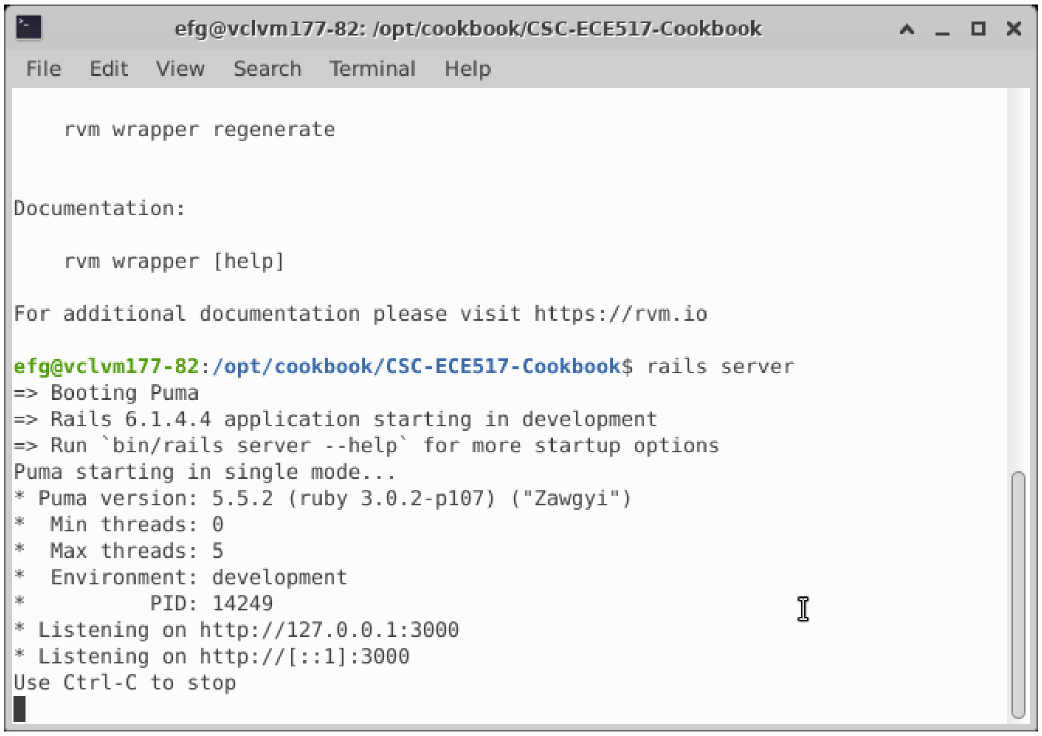
- the controller is **`users`**,
- the action is **`show`**, and
- the ID of the user is **`108`**.

Or, we might have something like
`http://www.etailer.com/store/add_to_cart/353`.

What do you think this would represent?

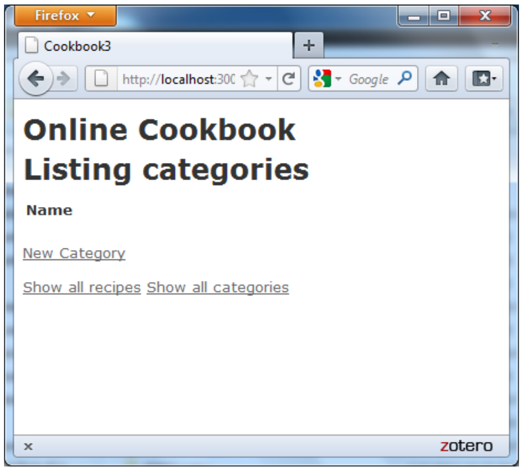
The Rails Cookbook Application

After we start the server, we see this series of messages:


efg@vclvm177-82: /opt/cookbook/CSC-ECE517-Cookbook
File Edit View Search Terminal Help

```
    rvm wrapper regenerate


Documentation:

    rvm wrapper [help]

For additional documentation please visit https://rvm.io

efg@vclvm177-82:/opt/cookbook/CSC-ECE517-Cookbook$ rails server
=> Booting Puma
=> Rails 6.1.4.4 application starting in development
=> Run `bin/rails server --help` for more startup options
Puma starting in single mode...
* Puma version: 5.5.2 (ruby 3.0.2-p107) ("Zawgyi")
*  Min threads: 0
*  Max threads: 5
*  Environment: development
*          PID: 14249
* Listening on http://127.0.0.1:3000
* Listening on http://[::1]:3000
Use Ctrl-C to stop
```

When we visit the application in our browser, we see this initial screen:



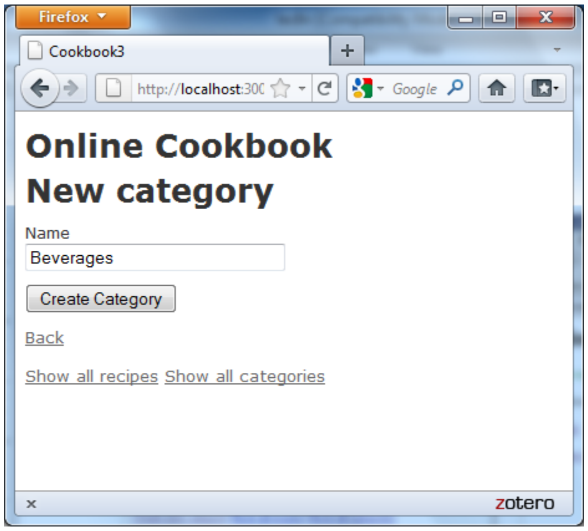
Online Cookbook
Listing categories

Name

New Category

Show all recipes Show all categories

Let’s create a category:



Online Cookbook

New category

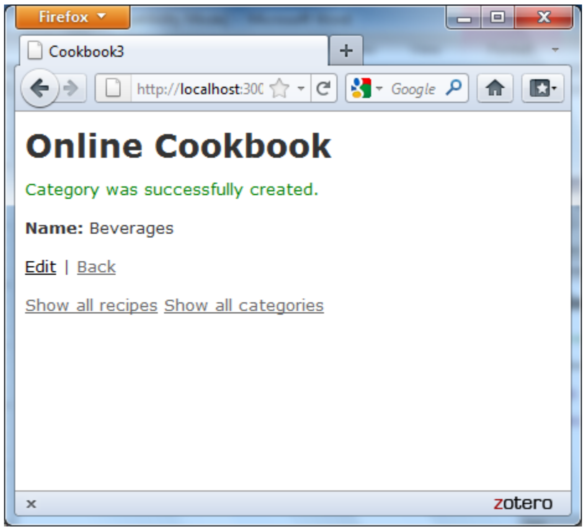
Name

Beverages

Create Category

Back

Show all recipes Show all categories



Online Cookbook

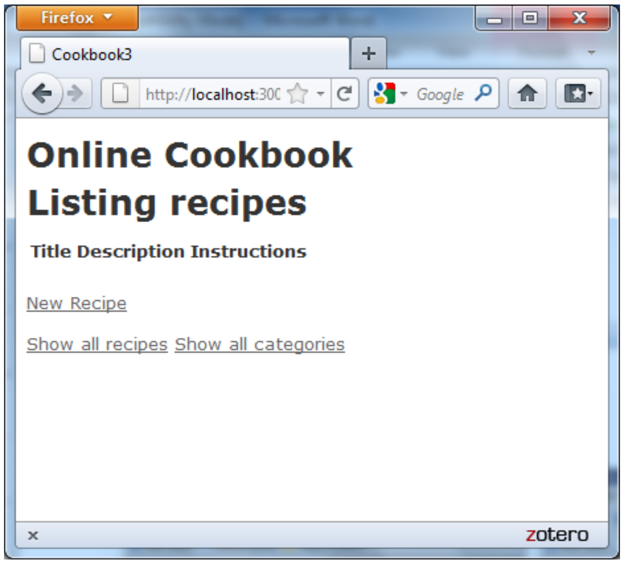
Category was successfully created.

Name: Beverages

Edit | Back

Show all recipes Show all categories

And a couple of beverages …



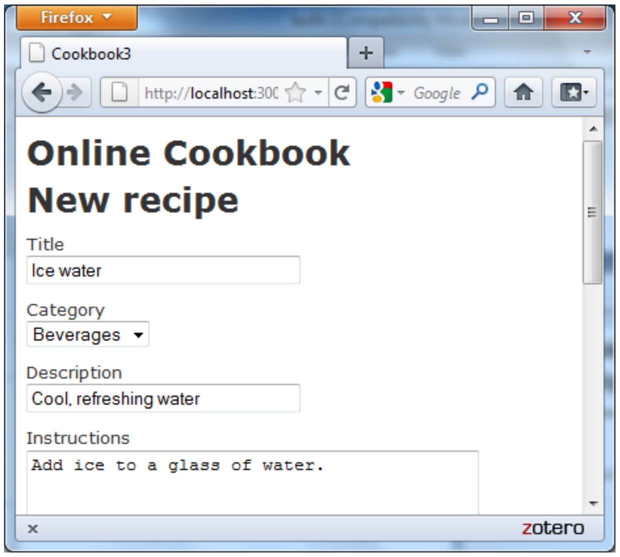
Online Cookbook

Listing recipes

Title Description Instructions

New Recipe

Show all recipes Show all categories



Online Cookbook

New recipe

Title

Ice water

Category

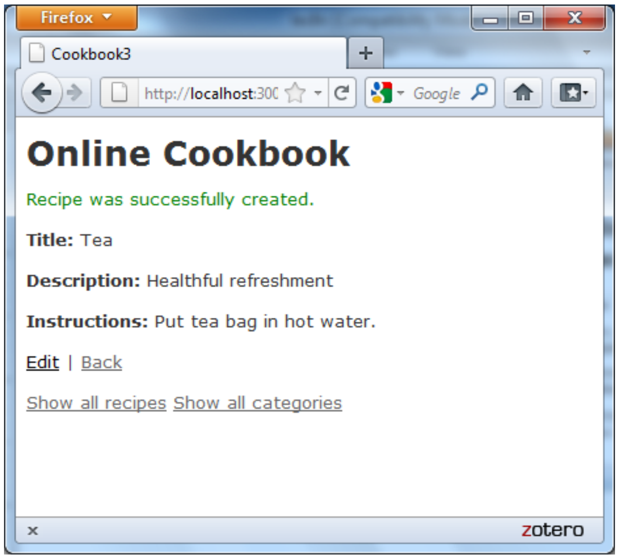
Beverages

Description

Cool, refreshing water

Instructions

```
Add ice to a glass of water.
```


Online Cookbook

Recipe was successfully created.

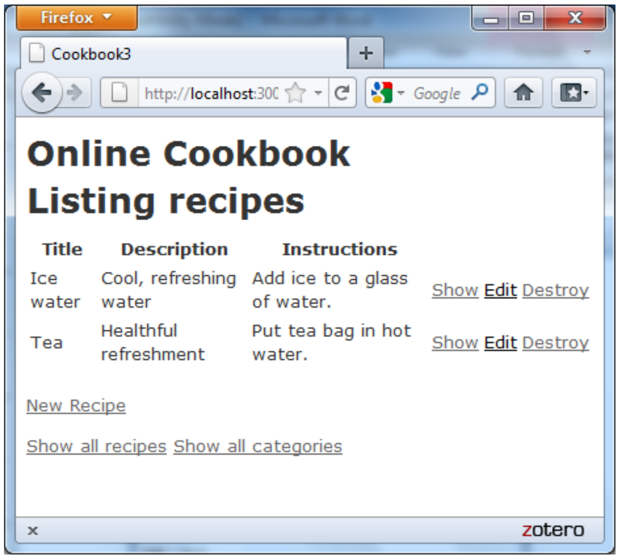
Title: Tea

Description: Healthful refreshment

Instructions: Put tea bag in hot water.

Edit | Back

Show all recipes Show all categories


Online Cookbook

Listing recipes

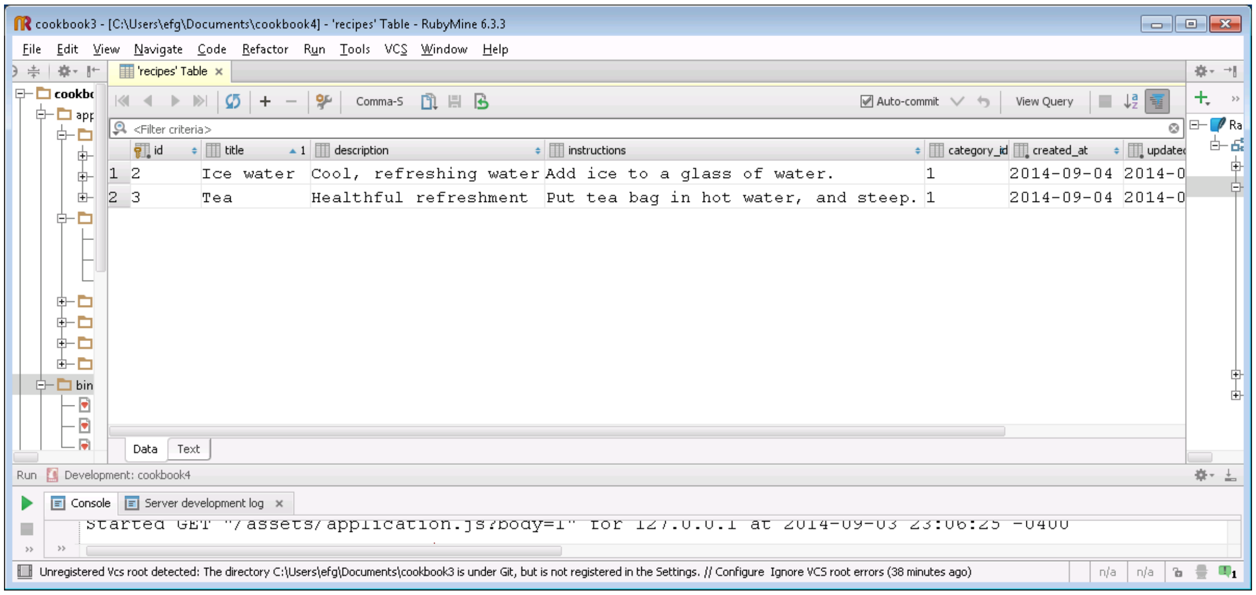
Title	Description	Instructions			
Ice water	Cool, refreshing water	Add ice to a glass of water.	Show	Edit	Destroy
Tea	Healthful refreshment	Put tea bag in hot water.	Show	Edit	Destroy

New Recipe

Show all recipes Show all categories

Now let’s go and look at what’s in the database:

- Go to `View → Tool Windows → Database`.
- Make sure that `Development: Cookbook` is added as a data source (see the RubyMine setup instructions for how to do this).
- Double-clicking on a table will show the records in that table.


cookbook3 - [C:\Users\efg\Documents\cookbook4] - 'recipes' Table - RubyMine 6.3.3

	id	title	description	instructions	category_id	created_at	update
1	2	Ice water	Cool, refreshing water	Add ice to a glass of water.	1	2014-09-04	2014-0
2	3	Tea	Healthful refreshment	Put tea bag in hot water, and steep.	1	2014-09-04	2014-0

```
Started GET "/assets/application.js?body=1" for 127.0.0.1 at 2014-09-03 23:06:25 -0400
```

Unregistered Vcs root detected: The directory C:\Users\efg\Documents\cookbook3 is under Git, but is not registered in the Settings. // Configure Ignore VCS root errors (38 minutes ago)

Notice the `created_at` and `updated_at` fields. These are automatically updated with the timestamp of the time that a row was created or updated.

Also notice the `category_id`. What do you think this is

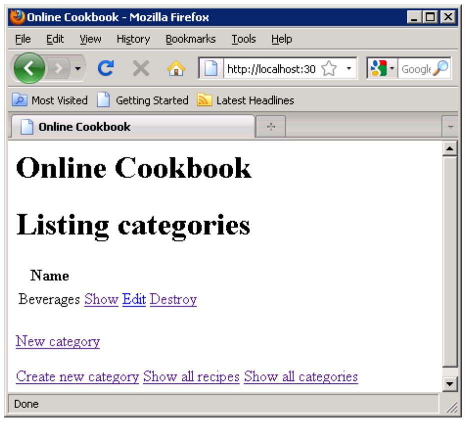
The controllers

Let's take a look at the code for **categories_controller.rb**.

```
class CategoriesController < ApplicationController
  # GET /categories
  # GET /categories.json
```

Note that it consists of set of actions, with each method implementing one of the actions.

```
 def index
    @categories = Category.all

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @categories }
    end
  end
```


Online Cookbook - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:30
Most Visited Getting Started Latest Headlines
Online Cookbook
Done

Online Cookbook

Listing categories

Name			
Beverages	Show	Edit	Destroy

New category

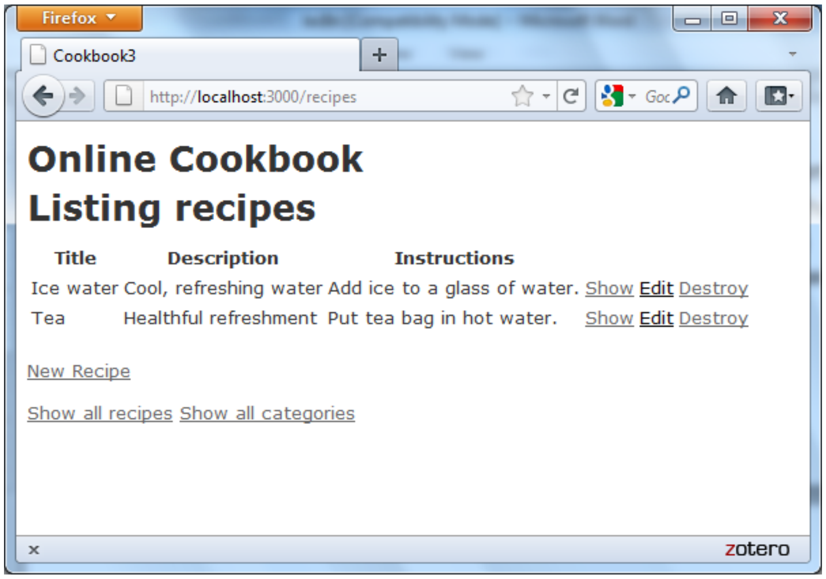
Create new category Show all recipes Show all categories

The statement inside the method does a database lookup of all categories, and assigns the resulting array to the **`@categories`** instance variable.

Without web-service support, that would be the whole method. The code that follows determines whether to respond with HTML (as when we are interacting with a user) or JSON (if we are returning an object).

What that says is, "if the client wants HTML, just send back the categories in HTML, but if the client wants JSON, return the list of categories in JSON format." The response format is determined by HTTP Accept header sent by the client.

Immediately after executing a method, e.g., **`index`**, the controller will render a template of the same name. This template is in the corresponding directory in the **`views`** folder. In this case, the template is **`views/categories/index.html.erb`**. In a few minutes, we will talk about **`.erb`** files.



Online Cookbook

Listing recipes

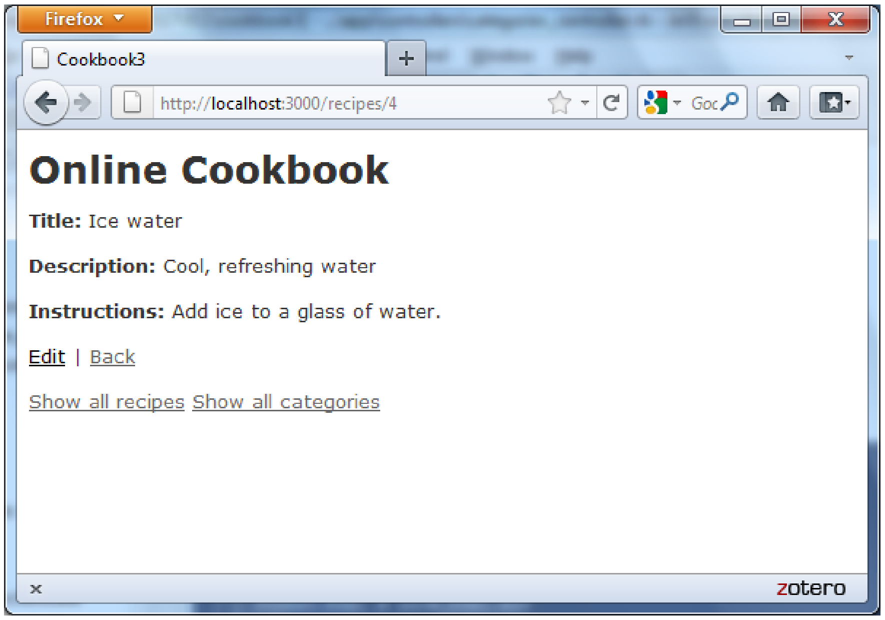
Title	Description	Instructions			
Ice water	Cool, refreshing water	Add ice to a glass of water.	Show	Edit	Destroy
Tea	Healthful refreshment	Put tea bag in hot water.	Show	Edit	Destroy

New Recipe

Show all recipes Show all categories

There is very little difference between the **`index`** method and the **`show`** method.

```
# GET /categories/1
# GET /categories/1.json
def show
  @category = Category.find(params[:id])

  respond_to do |format|
    format.html # show.html.erb
    format.json { render json: @category }
  end
end
```



Online Cookbook

Title: Ice water

Description: Cool, refreshing water

Instructions: Add ice to a glass of water.

Edit | Back

Show all recipes Show all categories

The **`show`** method looks for a category with a particular key. Its output is very basic (above). The formatting is different (the text “Listing categories” doesn’t appear, etc.) because the corresponding views are different (as we will see).

Look again at the difference between the assignment statements involving categories.

- **`index`** has **`@categories = Category.all`**
- **`show`** has **`@category = Category.find(params[:id])`**

Both of these methods do retrievals from the **`categories`** table in the db.

- The **`all`** method retrieves all records from the **`categories`** table and assigns the collection to a variable called **`@categories`**.

- The **`find(...)`** method retrieves a particular record, the record with the specified **`id`**.

This is the first use we have seen of *Active Record*.

Now, as an exercise, see if you can arrange this set of statements to form the **`index`** method of **`recipes_controller.rb`**.

Active Record

Our Ruby on Rails programs deal with *objects*, but they are mapped into relational databases.

There's a mismatch here—how are the database tables translated into objects, and how are objects created in the program saved to the db?

Ruby on Rails' solution is Active Record. In Active Record,

- Database tables correspond to Rails (model) classes.
- Database records (rows) correspond to Rails objects.

We can perform operations on tables by invoking class methods, as is done in both the RecipesController and the CategoriesController:

```
@recipes = Recipe.all

@categories = Category.all
```

Let's take a closer look at the **`find`** in the **`show`** method:

> **`Category.find(params[:id])`**

This illustrates two common features of Active Record calls.

- **`params`** is an object (a hash) that holds all of the parameters passed in a browser request.
- **`params[:id]`** holds the id, or primary key, of the object.

When you click a link for a specific category, the id of that category is passed in the **`params`** object, so that **`show`** can display that category.

Now, if you are running the application on your computer, you can test what happens if you delete a category that contains a recipe.

Method pairs

Next we have the **`new`** and **`create`** methods. Sounds like they might do the same thing …

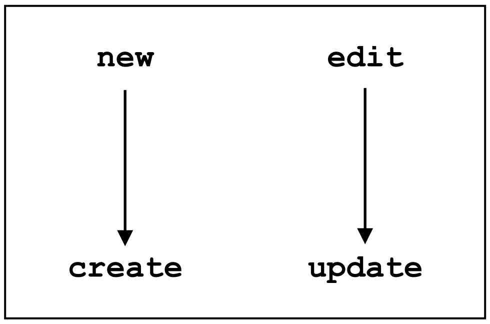
What's the difference between the two? Well, which one is called first?

The ______ method prepares the form for display; the ______ method processes the data that was entered and attempts to save it to the db.

```
  # GET /categories/new
  def new
    @category = Category.new

    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @category }
   end

  # POST /categories
  # POST /categories.json
  def create
    @category = Category.new(category_params)

    respond_to do |format|
      if @category.save
        format.html { redirect_to @category, notice: 'Category was successfully created.' }
        format.json  { render json: @category, status: :created, :location: @category }
      else
```

```
        format.html { render action: "new" }
        format.json  { render json: @category.errors, :status: :unprocessable_entity }
      end
    end
  end
```

There is a similar distinction between **`edit`** and **`update`**. Edit retrieves a table entry and displays it in a window. When the changes are submitted, **`update`** is invoked.


$$\begin{tikzcd}
\texttt{new} \arrow[d] & \texttt{edit} \arrow[d] \\
\texttt{create} & \texttt{update}
\end{tikzcd}$$

```
  # GET /categories/1/edit
  def edit
    @category = Category.find(params[:id])
  end

  # PUT /categories/1
  # PUT /categories/1.xml
  def update
    respond_to do |format|
      if @category.update_attributes(category_params)
        format.html { redirect_to @category, notice: 'Category was
successfully updated.' }
        format.json  { head :no_content }
      else
        format.html { render action: 'edit' }
        format.xml  { render json: @category.errors, status:
:unprocessable_entity }
      end
    end
  end

  # DELETE /categories/1
  # DELETE /categories/1.xml
  def destroy
    @category.destroy

    respond_to do |format|
      format.html { redirect_to categories_url }
      format.xml  { head :no_content }
    end
  end
```

The categories_controller is extremely similar to the recipes_controller. Let's take a look …

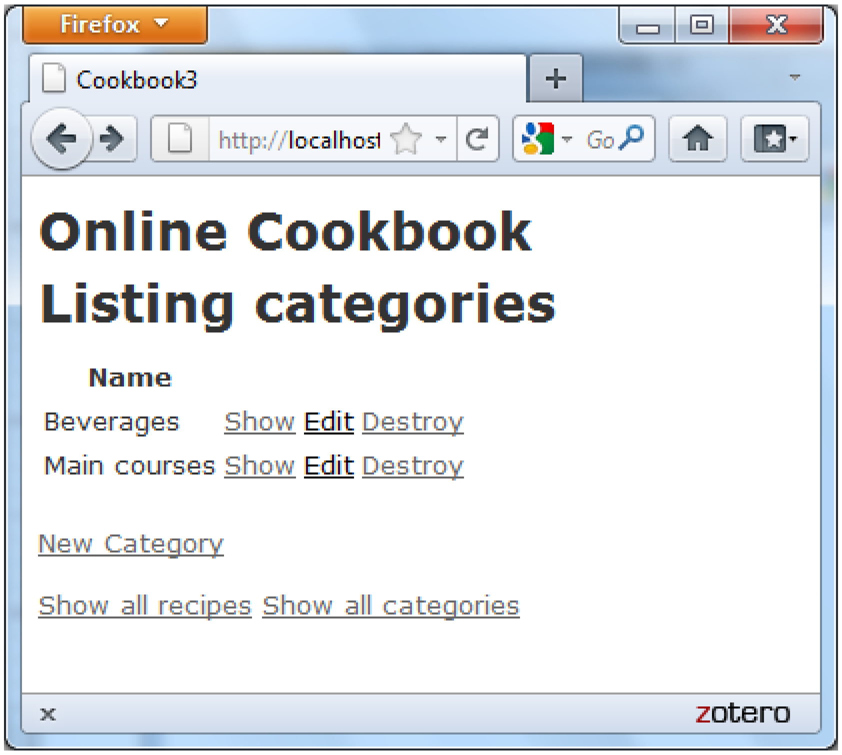
These controllers were generated by the Rails *scaffold* mechanism. We can use the scaffold to create a recipes table, with title, description, and instructions fields.

We can then proceed to exercise the application as we did before. But now we can change its functionality too.

Let's make a trivial change: Change "Show all recipes" and "Show all categories" to "List all recipes and "List all categories".

Where shall we make this change?

Well, if we click on "Show all categories", we get this screen.



Online Cookbook

Listing categories

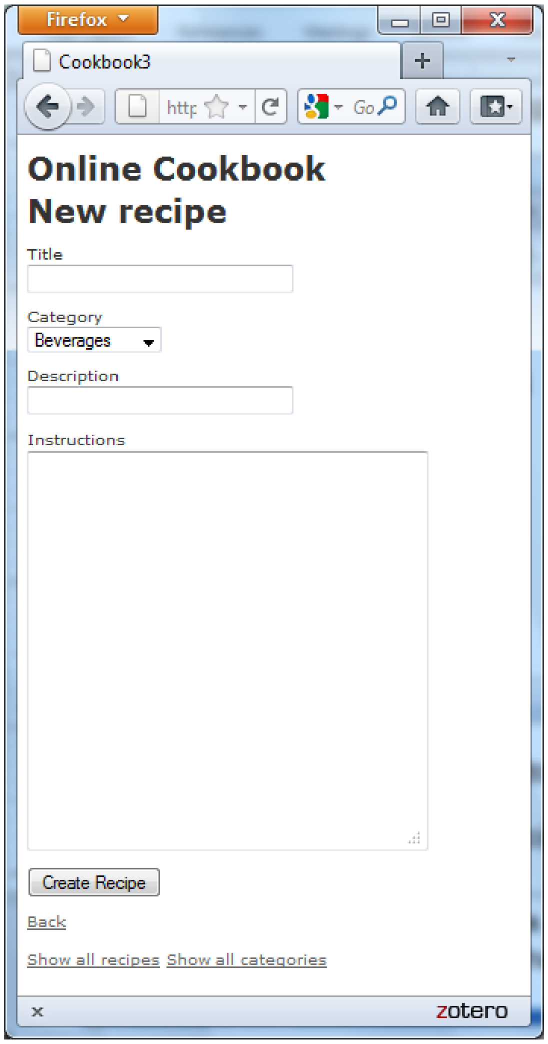
Name			
Beverages	Show	Edit	Destroy
Main courses	Show	Edit	Destroy

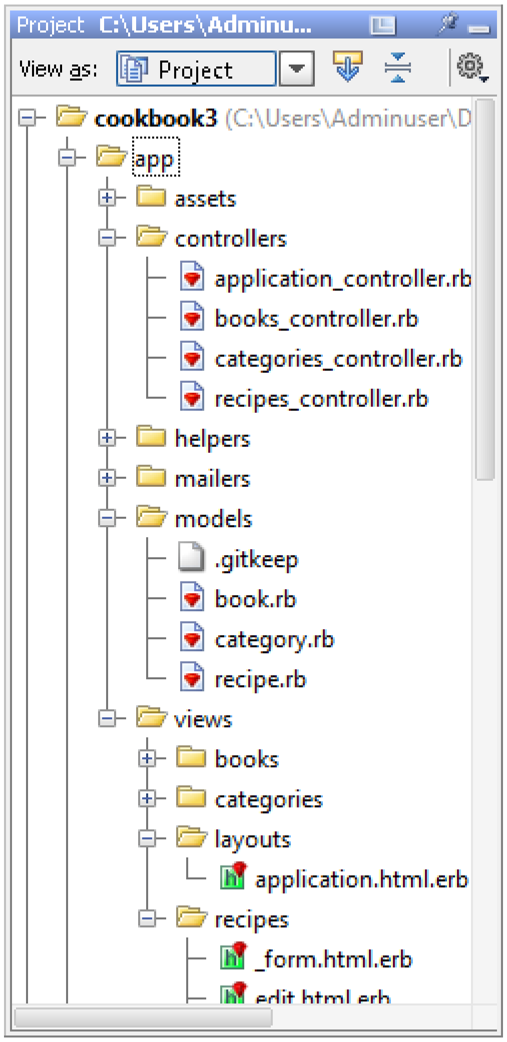
New Category

Show all recipes Show all categories

What do you notice about the bottom line?

And if we click on "Create new recipe", we get this screen:





```
cookbook3 (C:\Users\Adminuser\D
  app
    assets
    controllers
      application_controller.rb
      books_controller.rb
      categories_controller.rb
      recipes_controller.rb
    helpers
    mailers
    models
      .gitkeep
      book.rb
      category.rb
      recipe.rb
    views
      books
      categories
      layouts
        application.html.erb
      recipes
        _form.html.erb
        edit.html.erb
```

Again, the bottom line is the same. Let's take a look at the directory listing. Which file do you think contains that code?

This is an .html.erb file, the first time we've seen this type. What do you think it stands for?

Let's look at the code in this file.

```
<!DOCTYPE html>
<html>
<head>
  <title>Online Cookbook</title>
  <%= stylesheet_link_tag    "application", :media => "all" %>
  <%= javascript_include_tag "application" %>
  <%= csrf_meta_tags %>
</head>
<body>
<h1>Online Cookbook</h1>

<%= yield %>
<br><br>

<%= link_to "Show all recipes", recipes_url %> <%= link_to "Show all
categories", categories_url %>

</body>
</html>
```

This raises several questions.

- What gets invoked when we click on the “Show all recipes” link?

- What is the **`<%= yield %>`** for?

- What do “**`recipes_url`**”, “**`categories_url`**”, etc., mean?

Some actions, like deleting an object, are dangerous. Our app should—and does—guard against performing them by accident. What code could we add to do the same when updating a recipe?

Now, here are some review questions.

1. What URL do we type in to find the homepage of our cookbook application?
2. When we click on “Show all categories”, what URL will be taken to?

Routes

Routes are a way to take incoming requests from the web and redirect them to specific controllers and actions (actions == methods).

In the **routes.rb** file, you just declare

```
    resources :categories
```

and this generates routes for each of the methods for the categories_controller.

In the table below, the *path* is the tail of the URL that is visited.

When that path is visited, a particular controller is invoked. The *action* is the method of the controller that will be called.

HTTP Verb	Path	Controller#Action	Used for
GET	/categories	categories#index	display a list of all categories
GET	/categories/new	categories#new	return an HTML form for creating a new category
POST	/categories	categories#create	create a new category
GET	/categories/:id	categories#show	display a specific category
GET	/categories/:id/edit	categories#edit	return an HTML form for editing a category
PATCH/PUT	/categories/:id	categories#update	update a specific category
DELETE	/categories/:id	categories#destroy	delete a specific category

Some paths specify an id, which is the object (or database record) that is being acted upon.

Also, creating routes creates “helpers” …

Variable	Value	Example
categories_path	/categories	
new_category_path	/categories/new	
edit_category_path(:id)	/categories/:id/edit	edit_category_path(10) → /categories/10/edit
category_path(:id)	/categories/:id	category_path(10) → /categories/10

For each **`_path`** variable, there is a corresponding **`_url`** variable that contains the whole path, e.g., **`http://localhost:3000/categories/new`**

Test your knowledge of routes with this exercise.

The models

There are only two files in the model, one each for the tables in the application.

Let's take a look at them.

category.rb

```
class Category < ActiveRecord::Base
  has_many :recipes
  validates :name, :presence => true
end
```

recipe.rb

```
class Recipe < ActiveRecord::Base
  belongs_to :category

  validates :title, :presence => true
  validates :description, :presence => true
  validates :instructions, :presence => true
  validates :category, :presence => true
end
```

Many validations can be applied to fields of a model object. This Rails Guide describes them. Use it to answer these questions about validations that could be added to our Cookbook app.

The **`validates`** of Rails 3 can be replaced with a strong-parameter mechanism, starting with Rails 4.

This prevents mass assignment, which is a security hole.

Active Record associations

A relationship between two objects may be

- one-to-one (e.g., a course has a syllabus and a syllabus belongs to one course).

  -
    ```
    class Syllabus < ActiveRecord::Base
        belongs_to :course
        # …
    end
    ```

  -
    ```
    class Course < ActiveRecord::Base
        has_one :syllabus
        # …
    end
    ```
- one-to-many (e.g., a course has many assignments)

  -
    ```
    class Assignment < ActiveRecord::Base
        belongs_to :course
        # …
    end
    ```

  -
    ```
    class Course < ActiveRecord::Base
        has_many :assignments
        # …
    end
    ```
- many-to-many (e.g., a course has many students; students have many courses)

  -
    ```
    class Student < ActiveRecord::Base
        has_and_belongs_to_many :courses
    ```

```
        # …
    end
```

-
  ```
  class Course < ActiveRecord::Base
      has_and_belongs_to_many :students
      # …
  end
  ```

Rails calls these relationships associations.

You should be able to answer the following questions about these files.

1. What kind of relationship is there between recipes and categories?

2. Where is this relationship represented?

Earlier, we saw that deleting a category before deleting its recipes caused an error. Now, by reading Section 1 of Active Record Associations, you should be able to determine how to fix this problem.

Now, you can probably see how models, views, and controllers fit together.

The views

Now, let's take a look at the View code for the categories. We'll look at it line by line, which may obscure the flow, but if you have trouble, just look in your Cookbook folder for the uncommented code.

edit.html.erb

```
<h1>Editing category</h1>

<%= render 'form' %>

<%= link_to 'Show', @category %> |
<%= link_to 'Back', categories_path %>
```

The `form` refers to a partial named **_form.html.erb**.

_form.html.erb

```
<%= form_for(@category) do |f| %>
  <% if @category.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@category.errors.count, "error") %>
prohibited this category from being saved:</h2>

      <ul>
      <% @category.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :name %><br>
    <%= f.text_field :name %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

Compare this with **_form.html.erb** for recipes:

```
<%= form_for(@recipe) do |f| %>
  <% if @recipe.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@recipe.errors.count, "error") %> prohibited
this recipe from being saved:</h2>

      <ul>
      <% @recipe.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :title %><br>
    <%= f.text_field :title %>
  </div>
  <div class="field">
```

```
    <%= f.label :description %><br>
    <%= f.text_field :description %>
  </div>
    <div class="field">
      <%= f.label :category %><br />
      <%= select("recipe", "category_id", Category.all.collect{ |c| [
c.name, c.id] }) %>
    </div>
  <div class="field">
    <%= f.label :instructions %><br>
    <%= f.text_area :instructions %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

Explain the differences.

show.html.erb

This file has very basic functionality.

```
<p id="notice"><%= notice %></p>

<p>
  <b>Name:</b>
  <%= @category.name %>
</p>

<%= link_to 'Edit', edit_category_path(@category) %> |
<%= link_to 'Back', categories_path %>
```

Compare with **show.html.erb** for recipes.

```
<p id="notice"><%= notice %></p>

<p>
  <b>Title:</b>
  <%= @recipe.title %>
</p>

<p>
  <b>Description:</b>
  <%= @recipe.description %>
</p>

<p>
  <b>Instructions:</b>
  <%= @recipe.instructions %>
</p>


<%= link_to 'Edit', edit_recipe_path(@recipe) %> |
<%= link_to 'Back', recipes_path %>
```

new.html.erb

The only remaining view is **new.html.erb**. It doesn't illustrate much that we haven't seen before, so I'll ask you the questions (below).

```
<h1>New category</h1>

<%= render 'form' %>

<%= link_to 'Back', categories_path %>
```

1. Which controller is invoked when the form is submitted? Where is the code for this controller?

2. What says to print out a blank for the name of the category?