[Lesson 7.5: AJAX](#h.q3bk55cgx21w)

[Gems of the Day](#h.vhg12fshm5ln)

[What is AJAX?](#h.pk8ylwgveejv)

[Intro Exercise: Blog](#h.j92te4x1lhkv)

[Task App](#h.8m4xg815dlrn)

[Rails Composer](#h.tc8zpnalfa9z)

[Adding Tasks](#h.kej5s3e98edw)

[Switching to AJAX: Controller](#h.xiwv25rcwpol)

[New Form Using AJAX](#h.heehy87mumya)

[Dynamically Updating Index](#h.tyxxi511e2o5)

[Updating a Task](#h.vf8e1r4pkt0v)

[Deleting a Task](#h.ks956vveukg8)

[Beautification](#h.nbdvxnczammd)

[Datepicker](#h.xwylv6tywsxl)

[Layout](#h.il2qa24ep98k)

[Fancy Deleting](#h.ih4oksl6pz4h)

[Homework](#h.vakftzj5soki)

# 

# Lesson 7.5: AJAX

## Gems of the Day

* Rails Composer <https://github.com/RailsApps/rails-composer>
* Simple Form <https://github.com/plataformatec/simple_form>
* Bootstrap Datepicker Rails <https://github.com/Nerian/bootstrap-datepicker-rails> - even has a [sandbox](https://github.com/Nerian/bootstrap-datepicker-rails) to build your datepicker JS code

## What is AJAX?

AJAX stands for Asynchronous JavaScript and XML.

* ASYNCHRONOUS means that the client can request new pieces of information from the server at ANY TIME
* This means content on a page is updated without having to re-render the entire page, making it a “seamless” experience.

## Intro Exercise: Blog

Create a new rails project called ajax-sample inside your rails-practice folder.

$ rails new ajax-sample

$ cd ajax-sample  
$ rails g scaffold Post content:text  
$ rake db:migrate

Create a method inside your Posts controller called ajax

# posts\_controller.rb  
def ajax  
end

Create the corresponding ajax view and route.

# routes.rb  
root 'posts#ajax'

<!-- ajax.html.erb -->  
<h1>Ajax Testing 1, 2, 3</h1>

<!-- This div element will be replaced by actual posts. First, we just put a link to the JavaScript function that will do that behavior. -->

<div id="posts">

<a href="javascript:loadPosts();">This is where blog posts go</a>

</div>

<!-- Here is the JavaScript function that actually loads posts once it's initiated by clicking the hyperlink above. -->

<script type="text/javascript">

function loadPosts(){

// Go grab the JSON data which contains our posts. Rails creates the JSON version of our data for us (index.json.jbuilder).

$.getJSON("/posts.json", function(data){

// Put the post data in an HTML string to make it bold and italic

var html = "";

$.each(data, function (index){

// alert(data[index].content);

html += "<p><strong><em>" + data[index].content + "</em></strong><p>";

}); // close .each

// update our div with the content of the var html

$("#posts").html(html);

}); // close .getJSON

} // close function

</script>

Add a few posts (/posts), then go back to the root and see what happens. Notice that the entire page did not have to reload in order to return the data for the posts. This is AJAX coolness!

## Task App

### Rails Composer

Let’s use Rails Composer to create our new task app. From the readme, we know to run this command (changing the app name to what you want your app to be named):

rails new myapp -m https://raw.github.com/RailsApps/rails-composer/master/composer.rb

Windows users will probably have an SSL error - here is the fix:

<https://gist.github.com/fnichol/867550>

Here are the options I generally choose for an app like this - feel free to experiment. However, in the beginning, I recommend you use less of the functionality until you get comfortable working with each of these tools.

* Build a starter application? 3) Custom application (experimental)
* Web server for development? 4) Puma
* Web server for production? 1) Same as development
* Database used in development? 1) SQLite
* Template engine? 1) ERB
* Test framework? 1) None (use default minitest)
* Front-end framework? 2) Bootstrap 3.3
* Add support for sending email? 1) None
* Authentication? 2) Devise
* Devise modules? 1) Devise with default modules
* Authorization? 2) Simple role-based
* Admin interface for database? 1) None (add ActiveAdmin later)
* Use a form builder gem? 2) SimpleForm
* Prepare for deployment 1) none or 2) Heroku
* Add gem and file for environment variables? 3) Add application.yml with Figaro

Notice that in both secrets.yml and application.yml, you have some default roles set up. These provide the admin login details you would need to log in as an admin - your database is currently seeded with this fake user (eventually you want to change this otherwise anyone can guess).

Go ahead and run your server and take a look at the app. Commit your changes.

### Adding Tasks

Let’s add Tasks with a description and deadline (“task” is a reserved word in Rails):

rails g scaffold UserTask description:string due:date

rake db:migrate

Commit your changes, then run your server and add some data.

### Switching to AJAX: Controller

Let’s hijack our “new” link so that we can load a hidden form. Here we pass the remote: true option to disable the default Rails mechanism that would have otherwise navigated us to /tasks/new.

<%= link\_to 'New Task', new\_user\_task\_path, remote: true %>

Now, we are going to edit our controller:

1. First, let’s create a **before action** to use an all\_tasks method which will set an instance variable @user\_tasks = UserTask.all:  
   class UserTasksController < ApplicationController  
    before\_action :all\_tasks, only: [:index, :create, :update, :destroy]  
    before\_action :set\_user\_task, only: [:show, :edit, :update, :destroy]  
    …  
   private  
    def all\_tasks  
    @user\_tasks = UserTask.all  
    end
2. Since that is essentially the same code that is in the **index** method, we no longer need that method, so we can remove it from the file along with the **show** and **edit** methods, which are also blank. We are also going to delete the **destroy** methods since we don’t want to do those responses whenever a user updates or deletes.
3. Finally, let’s edit the **create** method to do UserTask.create rather than UserTask.new, and delete the response stuff:  
    def create  
    @user\_task = UserTask.create(user\_task\_params)  
    end

You should only have 3 non-private methods left - new, create, and update.

### New Form Using AJAX

Now, let’s go to the index view and put in a div that we will populate using AJAX, similar to the Google Maps div for the map canvas. This div will hold our task form, so we will give it an id of task-form:

<div id="task-form"></div>

In your stylesheet, set the display to none since we want it hidden when the page initially loads:

div#task-form {

display: none;

}

Now we need to add the JavaScript code to return that form. In your views/user\_tasks folder, create a new file called **new.js.erb**. In that file, we will tell JS to look for the div with id of task-form, and in that div render the form, then add a bit of animation:

$('#task-form').html("<%= j (render 'form') %>");

$('#task-form').slideDown(350);

This script essentially will take the place of the “render form” part of new.html.erb. Run your server and check it out. You can’t actually create a new task yet but you can see the form render immediately. Commit your changes.

Let’s make the form more user-friendly while we fix it. Check out the documentation for [Simple Form](https://github.com/plataformatec/simple_form). We already installed the gem with Rails Composer, and it already conveniently used the Simple Form syntax for our form. We just need to add the remote: true to the form loop:

<%= simple\_form\_for(@user\_task, remote: true) do |f| %>

### Dynamically Updating Index

We aren’t quite finished yet because when we hit submit, it will want to go to a create view. Upon creation, we want it to re-render our full list dynamically. Let’s edit the index to update the list with JavaScript. First we need the placeholder in **index.html.erb**:

<h1>Tasks</h1>

<%= link\_to 'New Task', new\_user\_task\_path, remote: true, class: 'btn btn-default' %>

<div id="task-form"></div>

<div id="tasks">

<ul>

<%= render @user\_tasks %>

</ul>

</div>

We should practice more partials, so we referenced a new partial that will return the description and due dates. Create a new file in that folder called **\_user\_task.html.erb**:

<li>

<%= user\_task.description %>

<%= user\_task.due %>

</li>

Now, let’s create the JavaScript that will populate the tasks. What do you think it should be called? “create”. Create a new file in the same folder called **create.js.erb** - this is where we will say to render the tasks in the list as well as animate the form away.

$('#tasks').html("<%= j (render @user\_tasks) %>");

$('#task-form').slideUp(350);

Run your server and check it out! Commit your changes.

### Updating a Task

First, let’s go to the controller and edit the update method to call for a new (as yet to be created) JavaScript method:

def update

@user\_task.update(user\_task\_params)

end

Where do we want a user to be able to edit a task? Probably in the task list. Let’s edit our helper for the user\_tasks:

<li>

<%= user\_task.description %>

<%= user\_task.due %>

<%= link\_to "Edit", edit\_user\_task\_path(user\_task), remote: true %>

</li>

Now, create an **edit.js.erb** that is exactly the same as new.js.erb. Similarly, create an **update.js.erb** that is exactly the same as create.js.erb. Run your server and check that it worked. Excellent! Commit your changes.

### Deleting a Task

What do you think we need to change? (destroy method in controller, link in partial, js method) We need to call a new JavaScript method in the controller:

def destroy

@user\_task.destroy

end

And, we need to add a delete link to the \_user\_task partial:

delete

And create **destroy.js.erb** which will simply re-render the list of tasks:

$('#tasks').html("<%= j (render @user\_tasks) %>");

Run your server and test that it worked. So awesome - we can actually do all tasks from this one page without having to re-render the entire page! Commit your changes.

## Beautification

Let’s make our app prettier and more like real apps.

First, go to your **scaffolds** style sheet and delete everything. Notice how your page changes. You may want to do this in your other apps as well.

We already have Bootstrap in our app, courtesy of Rails Composer, but we still need to set up some **structure**. Go to your application.html.erb and add a container, row, and columns:

<main role="main">

<div class="container-fluid">

<div class="row">

<div class="col-sm-6 col-sm-offset-3">

<%= render 'layouts/messages' %>

<%= yield %>

</div>

</div>

</div>

</main>

### Datepicker

Let’s go ahead and add our [datepicker gem](https://github.com/Nerian/bootstrap-datepicker-rails) and bundle:

gem 'bootstrap-datepicker-rails'

Now we need to update some configurations. Add this line to app/assets/stylesheets/application.css:

\*= require bootstrap-datepicker3

Add this line to app/assets/javascripts/application.js

//= require bootstrap-datepicker

Now, go play in the sandbox to build the code with your preferred options:

<http://eternicode.github.io/bootstrap-datepicker/>

Once you’re satisfied, let’s go ahead and put the placeholder in our \_form (and optimize it a bit) - the most important part is the “as string” (See Simple Form documentation):

<div class="form-inputs">

<%= f.input :description, as: :text, placeholder: "Make groceries" %>

<%= f.input :due, as: :string, placeholder: "Today!" %>

</div>

We use “as string” so that the input type will be changed to text which is what the datepicker element needs. If you run your server and inspect element, you will notice that Simple Form automatically gives that form input an id of “user\_task\_due”, so we will just user that id for our JavaScript.

Now let’s create a partial that will hold our code for the actual datepicker - put **\_datepicker.js.erb** in the same views/user\_tasks folder. Copy your sandbox code, but change the id that it is looking for to the id of our input element, and wrap it inside a document ready function like below. Note that you also must have the format set like below otherwise Rails won’t save the date.

$(document).ready(function(){

$('#user\_task\_due input').datepicker({

autoclose: true,

todayHighlight: true,

toggleActive: true,

format: "yyyy-mm-dd",

orientation: "top auto"

});

});

Now we need to call this partial in our new and edit js.erb files:

$('#task-form').html("<%= j (render 'form') %>");

$('#task-form').slideDown(350);

<%= render 'datepicker' %>

Run your server and test it out!

### Layout

We can order our tasks by updating the controller:

private

# Set @all\_tasks

def all\_tasks

@user\_tasks = UserTask.order("due")

end

Let’s go ahead and add the Font Awesome gem, then bundle:

gem "font-awesome-rails"

Then add this to your application.css in the top area with similar require statements:

\*= require font-awesome

Now, take the row and column divs out of application.html.erb and instead make 2 columns in our task index, and use a plus icon in the add task button (add a top margin to the Add task button in your css):

<div class="row">

<div class="col-sm-4">

<%= link\_to new\_user\_task\_path, remote: true do %>

<button class="btn btn-default add-task"><%= fa\_icon "plus" %> Add Task</button>

<% end %>

<div id="task-form"></div>

</div>

<div class="col-sm-8">

<h1>Tasks</h1>

<div id="tasks">

<%= render @user\_tasks %>

</div>

</div>

</div> <!-- row -->

Now let’s update the user\_task partial. See also Bootstrap [wells](http://getbootstrap.com/components/#wells):

<div class="well">

<%= user\_task.description %>

<span class="pull-right">

<%= user\_task.due %>

<%= link\_to edit\_user\_task\_path(user\_task), remote: true do %>

<%= fa\_icon 'pencil' %>

<% end %>

<%= link\_to user\_task, remote: true, method: :delete, data: { confirm: 'Are you sure?'} do %>

<%= fa\_icon 'trash-o' %>

<% end %>

</span>

</div>

### Fancy Deleting

What if we wanted to delete a task just by clicking on it? First, let’s put the description inside it’s own span:

<div class="well">

<span class="task">

<%= link\_to user\_task.description, user\_task, remote: true, method: :delete, data: { confirm: 'Are you sure?'} %>

</span>

<%= link\_to edit\_user\_task\_path(user\_task), remote: true do %>

<%= fa\_icon 'pencil' %>

<% end %>

<span class="pull-right">

<%= user\_task.due %>

</span>

</div>

Now, let’s warn the user by showing a strike-through when hovering on a task by adding this css:

a:hover {

text-decoration: none;

}

.well .task a:hover {

text-decoration: line-through;

}

## Homework

1. Add the stamp gem to format the date prettier in the view. Also research how we could make that date look prettier in the form itself. You can find some clues in [this stackoverflow post](http://stackoverflow.com/questions/7181220/rails-3-jquery-date-picker-date-not-saving-to-database).
2. Add a cancel button to the new and edit forms.
3. If multiple tasks fall on the same date list the date once with each task underneath.
4. Push to GitHub.

Optional Add Ons:

1. Add a condition to check if there are any tasks. If none are listed display a silly image and instruct the user to add a task!
2. Add validation and get error messages to print to the page.
3. Add the pagination gem
4. Push to Heroku