

## Views & Integration

---

### Background

---

We talked about views, and about CSS/Javascript integration in class ... this lab is meant to give you an opportunity to explore and practice some of what we talked about :)

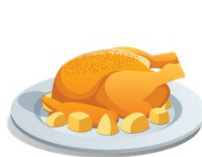
I have prepared a starter webapp for lab 5. You are going to enhance it.

There are two parts to this lab: A) the basics and B) a challenge. You only need complete part A to get a good mark. Part B is specifically for those looking for more of a challenge, and I suggest you only tackle it after successfully completing part A.



### Order #6 (5.95)

Click on a menu item below to add it to your order, or [Checkout](#)



Turkey (5.95)



Bubbly (14.50)



Cheese (2.95)



Hot Dog (6.90)



Coffee (2.95)



Donut (1.25)

---

## The Starter Webapp

---

I used the Lab 4 solution as the starting point for this lab, and added a minimal formfield helper and caboose, which I showed you in class on the 8<sup>th</sup>. If you are comfortable with your lab 4 solution, you can integrate the the starter with it, copying the missing pieces.

Work through the lab, and submit your zipped project to the lab 5 D2L dropbox, with a readme embedded.

---

## What Do You Do?

---

Using the same database as last week, we are going to add some menu maintenance ability. Much of what you will do is contrived, but is suitable as an excuse to work with different view techniques, which is the purpose of the lab.

You will add a new controller, “admin”, with a number of methods to display menu items and process changes to them. The rest of the webapp isn't affected.

Do not autoload any of the helpers or libraries used, to avoid name conflicts.

Without further ado...

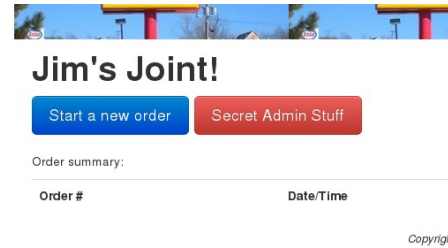
*ps – For better or worse, I have included the step number in the suggested method and view names in the following. If you would prefer more meaningful names, go for it!*

# Part A: Add an Admin Capability :)

## 1. Admin Menu List

Add a link (text or image) to the homepage, to your “admin” controller.

Build your admin controller, with the default method showing a list of all menu items, in a table. This will be very similar to the existing homepage controller and view, except showing the menu items.



### Jim's Joint Administration!

Our Menu List

Code	Name	Description	Picture
1	Cheese	Leave this raw milk, beefy and sweet cheese out for an hour before serving and pair with pear jam.	
2	Turkey	Roasted, succulent, stuffed, lovingly sliced turkey breast	
6	Donut	Disgustingly sweet, topped with artery clogging chocolate and then sprinkled with Pixie dust	
8	Hot Dog	Pork trimmings mixed with powdered preservatives, flavourings, red colouring and drenched in water before being squeezed into plastic tubes. Topped with onions, bacon, chili or cheese - no extra charge.	
10	Bubbly	1964 Moet Charmon, made from grapes crushed by elves with clean feet, perfectly chilled.	
11	Ice Cream	Combination of decadent chocolate topped with luscious strawberry, churned by gifted virgins using only cream from the Tajima strain of wagyu cattle	
21	Coffee	A delicious cup of the nectar of life, saviour of students, morning kick-starter; made with freshly grounds that you don't want to know where they came from!	
25	Burger	Half-pound of beef, topped with bacon and served with your choice of a slice of American cheese, red onion, sliced tomato, and Heart Attack Grill's own unique special sauce.	

Copyright © 2014, Me.

## 2. Menu List Using Nested View Fragments

Add a link (text or image) to your admin page, linking to “admin/list2”. Step 1 handled iteration in the view, whereas we will now handle it inside the controller.

Your list2 method needs to get the list of menu items and iterate over them, building a table row for each menu item using the “listitem2” view fragment, and adding that HTML to a temporary variable “themeat”. The consolidated view, list2, will then have the table setup and then substitute for “themeat” to provide the table rows.

The result should look the same as step 1, except we arrived at it differently.



### Jim's Joint Administration

Our Menu List [Show Editable Items!](#)

Code	Name	Description
1	Cheese	Leave this raw milk, beefy and sweet chee



### Jim's Joint Administration (view 2)!

Our Menu List #2

Code	Name	Description	Picture
1	Cheese	Leave this raw milk, beefy and sweet cheese out for an hour before serving and pair with pear jam.	
2	Turkey	Roasted, succulent, stuffed, lovingly sliced turkey breast	
6	Donut	Disgustingly sweet, topped with artery clogging chocolate and then sprinkled with Pixie dust	
8	Hot Dog	Pork trimmings mixed with powdered preservatives, flavourings, red colouring and drenched in water before being squeezed into plastic tubes. Topped with onions, bacon, chili or cheese - no extra charge.	
10	Bubbly	1964 Moet Charmon, made from grapes crushed by elves with clean feet, perfectly chilled.	
11	Ice Cream	Combination of decadent chocolate topped with luscious strawberry, churned by gifted virgins using only cream from the Tajima strain of wagyu cattle	
21	Coffee	A delicious cup of the nectar of life, saviour of students, morning kick-starter; made with freshly grounds that you don't want to know where they came from!	
25	Burger	Half-pound of beef, topped with bacon and served with your choice of a slice of American cheese, red onion, sliced tomato, and Heart Attack Grill's own unique special sauce.	

Copyright © 2014, Me.

### 3. Crude Item Editing

For the list2 view, add a column with an edit link for the menu item on that row. The link should be “/admin/edit3/#”, where “#” is the menu item number.



## Jim's Joint Administration (view 2)!

Our Menu List #2

Code	Name	Description	Picture
1	Cheese	Leave this raw milk, beefy and sweet cheese out for an hour before serving and pair with pear jam.	 <a href="#">Edit</a>
2	Turkey	Roasted, succulent, stuffed, lovingly sliced turkey breast	 <a href="#">Edit</a>
6	Donut	Disgustingly sweet, topped with artery clogging chocolate and then sprinkled with Pixie dust	 <a href="#">Edit</a>
8	Hot Dog	Pork trimmings mixed with powdered preservatives, flavourings, red colouring and drenched in water before being squeezed into plastic tubes. Topped with onions, bacon, chili or cheese - no extra charge.	 <a href="#">Edit</a>

Add an edit3 method, which will build an edit view, edit3, for the specified menu item, using only raw HTML. The form action should link back to “/admin/post/#”, again with the # being the menu item number.

The item number in the presented form should not be editable.

If you are really unhappy about the appearance, add some <br/> elements to make it look better, even if only marginally.


Something else to note: without a better tie-in between the model and the form, there is no easy way to flag the current item category properly.

You will need a post3 method, which makes sure that the name and description are present, that the price is numeric, and that the category is one of the allowed values. The menu item image should not be editable; just displayed, scaled if you wish.

If the changes are ok, update the record and redirect to the list2 menu.

If there are errors, display them in a fashion similar to what we did in lab 2.

Without a data transfer buffer, any changes are lost if there are errors ... that is a problem for an upcoming lab :)



## Jim's Joint Maintenance


Item Code:


Name:

Description:

Price:

Category:

Picture: 



## Jim's Joint Maintenance

An item has to have a name!

Item Code:

Name:

Description:

### 3a. Error Display

Hmm – how did I get that nice error display? I modified the template, adding an {errormessages} substitution field. I modified the base controller to build that for me, if there were any errors in the array already built for that purpose, on line 26 of MY\_Controller: \$this->errors = array();

The changes to MY\_Controller:

At line 37, I added logic to set the errormessages view parameter...

```
$this->data['errormessages'] = $this->scold();
```

Before the end of the class, I added a new method (called above) to do this:

```
/**
 * Build a nice display of any error messages
 *
 */
function scold() {
    $result = '';
    if (count($this->errors) < 1) {
        $this->data['alerting'] = '';
    } else {
        $this->data['alerting'] = 'alert alert-error';
        foreach ($this->errors as $msg) {
            $result .= $msg . '<br/>'; // the break should
be a constant somewhere
        }

    }
    return $result;
}
```

And in the template, between the title and content, I added a div to display any messages. The “alerting” class is left empty if no messages, and set to “alert alert-error” in the scold method above. Bootstrap makes it show up nicely :)

```
<h1>{title}</h1>
<div class="{alerting}">
    {errormessages}
</div>
{content}
```

#### 4. Pretty Item Editing

Use the form field helper to build the presented HTML for a field, instead of the hard-coded HTML.

This means building an “fxxx” view parameter for the edit4 view, for the “xxx” field. The edit4 view will be loosely based on the edit3 one.

Submit the results to the post4 method for handling, with the same guidelines as for the last step.

## 5. Integrate the Caboose Library for the Price

Clone your formfield helper, to formfield2 helper, and add a new function to make a currency input field. We will need a jquery plugin to handle the smart field manipulation, and the view fragment will need to be styled to work with it. Our Caboose will manage the HTML code needed, per the example in class.

Do not use the HTML5 number input form field!

I recommend one of the following widgets:

<https://github.com/dubroe/bootstrap-money-field>

<http://www.jqueryscript.net/form/jQuery-Currency-Input-Filed-Mask-Plugin-maskmoney.html>

<http://plentz.github.io/jquery-maskmoney/>

<http://plugins.jquery.com/autoNumeric/>

<http://plugins.jquery.com/df-number-format/>



## 6. Add An Image Uploader

Add a jquery image uploader widget to your caboose and appropriate asset folders.

Note: this **could** be the image uploader provided in the starter, but there are so many more useful jquery ones out there!

We can now add an additional field to the form (now edit6 and post6), to prompt for a replacement image for a menu item.

You will need to change the form to support file uploading, with an additional property ...  
`enctype="multipart/form-data"`.

I suggest replacing a menu item's current image, so the `move_uploaded_file` will need the current folder and filename. If you want to allow a differently named or typed image, then the uploaded one will need to be moved to the proper folder, named properly, and the "picture" property of a menu item will need that new name. I strongly suggest removing spaces and suspect characters in the uploaded filename automatically.

## 7. Process Uploaded Image

Edit7 & post7 :)

When an image is uploaded, use the CI image manipulation library to resize the image to be exactly 256x256, so it plays nicely with the other menu images. If the uploaded image isn't square, you have the choice of distorting it or cropping it (eg top-left only).

## 8. Make This Into a Plugin

Edit8 & post8, phew!

We have a helper (formfields), a library (caboose) and a bunch of view fragments (views/\_components, \_fields).

Let's put these together into a “package”, named at your discretion.

The package would go inside application/third\_party.

It can't be autoloaded, as it would conflict with the existing helpers and library, so the add\_package\_path call will have to be made inside your edit8 or post8 methods.

---

## Part B: Plugin

---

Tailor the plugin from the last step in Part A to work with your favorite non-Bootstrap CSS framework, and with several different widgets from the ones we have worked with. These could be a time picker, province picker, format-controlled field (eg postal code), and so on. Do not include the Caboose library, but provide a Caboose configuration for your plugin in its config/config.php.

This is for field-level formatting, not for layout styling. This means that your chosen framework will have to get along with Bootstrap.

---

## Marking Guideline

---

Part A has eight tasks, worth 1 mark each.

Part B is worth 2-3 marks, depending on the elegance of your solution. Simply taking a stab at part B, without solving the challenge, won't get you any marks!

*Commenting:* add comments sufficient to convince the reader that you know what you are doing. Inappropriate/inadequate comments will cost you up to two marks off.