

# Working With CodeIgniter Models

## tutorial #ci-normal02

**James L. Parry**  
**B.C. Institute of Technology**

---

## Tutorial Goals

---

We talked about models in class ... this tutorial is meant to give you an opportunity to explore and practice a bunch of what we talked about :)

I have prepared a starter webapp - a mini "menu ordering system". It has controllers and views, but is incomplete - lacking models, which you are to provide.

---

## Preparation

---

I have prepared a starter webapp. The lab 4 starter, Jim's Joint, is meant to handle ordering in a small restaurant. I have build some database tables for this (menu and then orders & order items), as well as some minimal controllers and views.

The webapp is "broken" as delivered, and will not run properly until you fix it :)

Fork the github project, and clone it locally to work with, the same as you have done with the previous tutorial.

---

# The End Result

---

This is what the end result should look like, after fixing!



## Order # 12 (0)

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 (5.90)



Coffee (2.95)



Donut (1.25)

---

# Database Setup

---

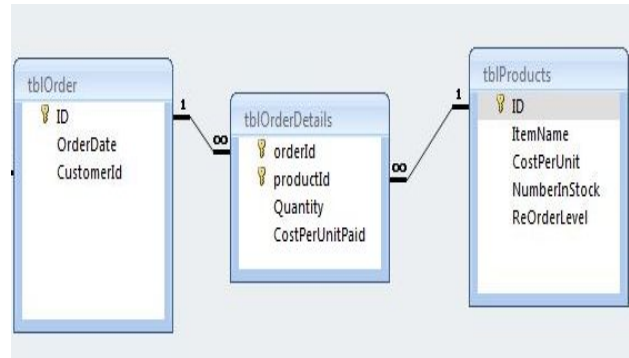
You will need to setup your webapp's database, using the supplied script, `data/comp4711-lab04-setup.sql`. Please name the database "comp4711", so that your webapp will work without additional setup when I run it on my system.

As part of the tutorial, you will then build models for each of the tables contained in it.

# Database Design

The RDB data provided is a pretty standard associative entity, very much like the diagram to the right. In our case, OrderItems is the associative entity/table connecting Menu and Orders.

In an O-O world, an ideal implementation would have an Order class, containing a collection of OrderItem objects. If we did that, the focus would end up more on good O-O practices rather than how you work with models inside a framework. The approach that I ended up taking is simple, but defensible.



## Data Structure - Menu Table

Field	Description	Notes
code	Unique menu identifier	Primary key
description	Menu item description	Used for alt text when image displayed
price	Price per item	Dollars & cents
picture	Name of item picture	Pictures in /assets/images
category	Menu category	m=meal, d=drink, s=sweet

## Data Structure - Orders Table

Field	Description	Notes
num	Order #	Primary key
date	Date/time order placed	Set when created, updated on checkout
status	Order status	a=open, c=complete, x=cancelled
total	Order total	Stored on checkout

---

## Data Structure - Orderitems Table

---

Field	Description	Notes
order	Order #	Part 1 of composite primary key
item	Menu item code	Within the order, part 2 of composite key
quantity	Quantity	Positive

---

## Model Implementation

---

You can use the database and active record classes built into CodeIgniter to complete this (using the CodeIgniter user guide for details), or you can build on the starter models in `core/MY_Model`. The former approach will get you deeper into the CodeIgniter "way", but the latter approach is the easier one.

The base model supplied with the starter app provides for single key tables (`MY_Model`) as well as composite key tables (`MY_Model2`).

---

## Controller Expectations

---

There are two controllers: `Welcome` and `Order`. The welcome page displays a summary of the orders handled so far, with a link to the order page to deal with order handling. Methods are provided in the `Order` controller to create a new order, to add an item to an order, and to "checkout". The checkout prompts for confirmation and takes the user back to the homepage if appropriate.

Each of the controller methods has comments and pseudo-code sufficient to guide you in the intended use of the model classes you need to build.

---

## What Needs Fixing?

---

There are seven steps to complete, the way I have broken the problem down. Each is described on following slides.

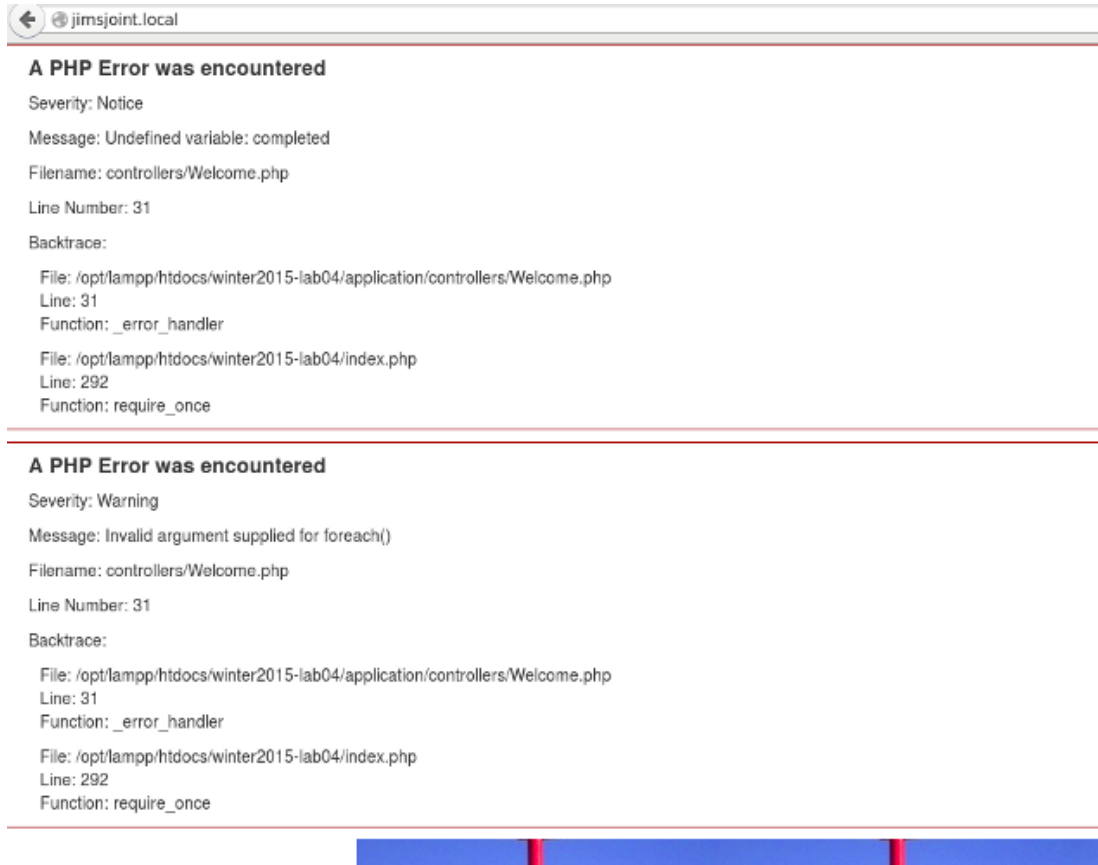
1. Configuration
2. Homepage
3. Handle a new order
4. Handle order display
5. Handle order additions
6. Handle checkout
7. Handle completion

---

# The Starting Point

---

The webapp is broken when you start, looking like this...



---

## 1. CONFIGURATION

---

Fix the config/autoload and config/database files.

config/autoload.php needs to load the database library. You can also load your three models here, or you can load them in the constructors of any controllers that need them.

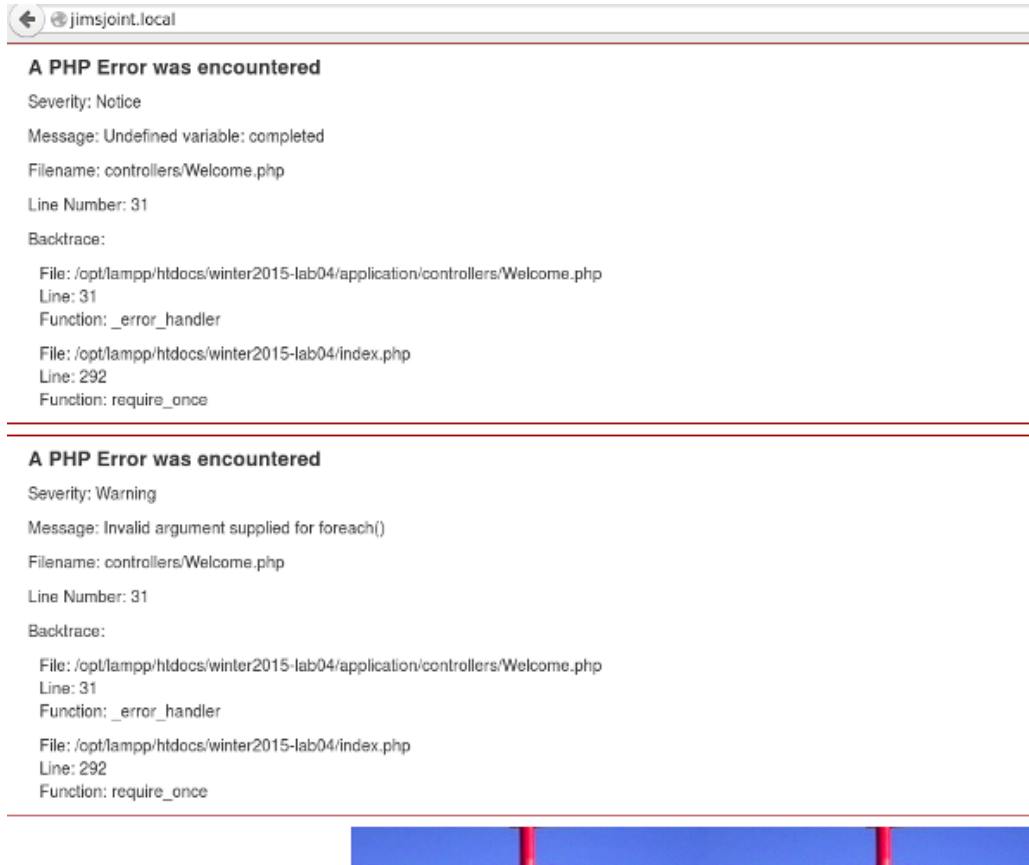
config/database.php needs to reference the appropriate database name, and MySQL username/password to work with them. If you are using this at home, or elsewhere, with a different username/password, then add a configuration subfolder with your settings, and "git ignore" it so it is not part of your repository. That would technically make your config config/development/database.php

---

# Post Configuration

---

There is no apparent difference after fixing your configuration...



---

## 2. HOMEPAGE

---

Fix the default controller. It needs to get completed order data from somewhere.

controllers/Welcome.php ... the loop at line 31 expects a variable \$completed to contain the order data, as an associative array. Completed orders can be distinguished by their status being 'c'.

# Fix the Homepage

Put this together, and the "fix" is to assign  
`$this->orders->some('status','c')`  
to `$completed`, if you are using the provided base models. I determined this by inspecting the `Active_record` interface at the top of `core/MY_Model.php`.

```
function index() {  
    $this->data['title'] = 'Jim\'s Joint!';  
    $this->data['pagebody'] = 'welcome';  
  
    // Get all the completed orders  
    $completed = $this->orders->some('status','c');  
  
    // Build a multi-dimensional array for reporting  
    $orders = array();
```

## Post Step 2

Your homepage should now look a bit different, showing any orders. You might not have any orders, if you are just getting underway.

Jim's Joint!

Start a new order

Order summary:

Order #	Date/Time	Amount
2	2014-10-09 03:27:44	38.84
3	2014-10-09 03:27:56	29.35
5	2014-10-09 03:29:17	14.50

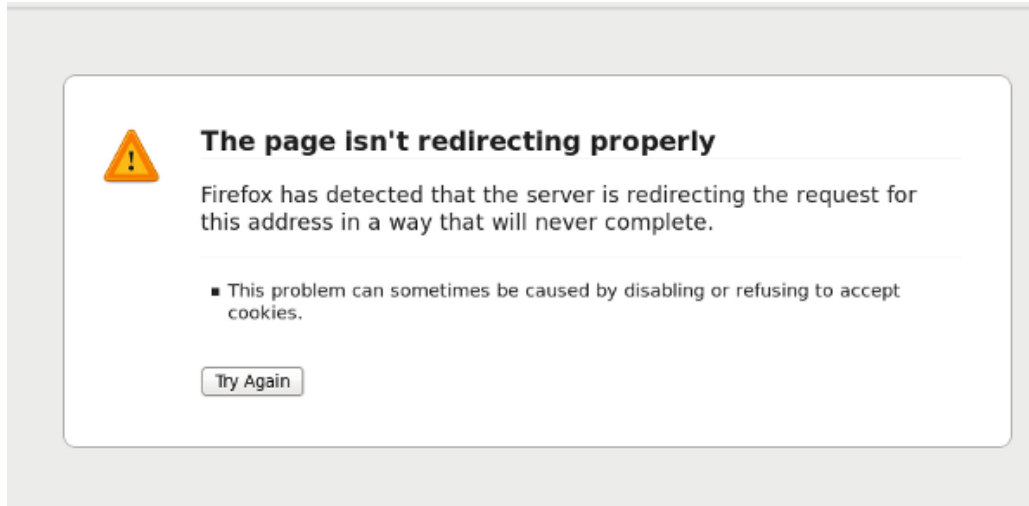
Copyright © 2014-2015, Me.

---

## Pre Step 3

---

If you try the next logical thing, "Start a new order", you will be presented with a nasty browser error page. This happens because we need to create a new order object before we can start working with it.



---

## 3. HANDLE A NEW ORDER

---

You need to fix `controllers/Order::neworder()`, creating a new order record with an order number one higher than the last used.

`Active_record::highest()` returns the highest key used in a model/table.  
`$this->orders->highest()` gives the last order # used.  
I suggest setting `$order_num` to that plus 1.

`Active_record::create()` creates a new record.  
`$this->orders->create()` hence creates a new object, with all the fields from the Orders table, initialized to blank.  
Set the order properties properly (number, current date, status), then save the new order object (add it to the orders model).



# Fix neworder()

`Active_record::create()` creates a new record.

`$this->orders->create()` hence creates a new object, with all the fields from the Orders table, initialized to blank.

Set the order properties properly (number, current date, status), then save the new order object (add it to the orders model).

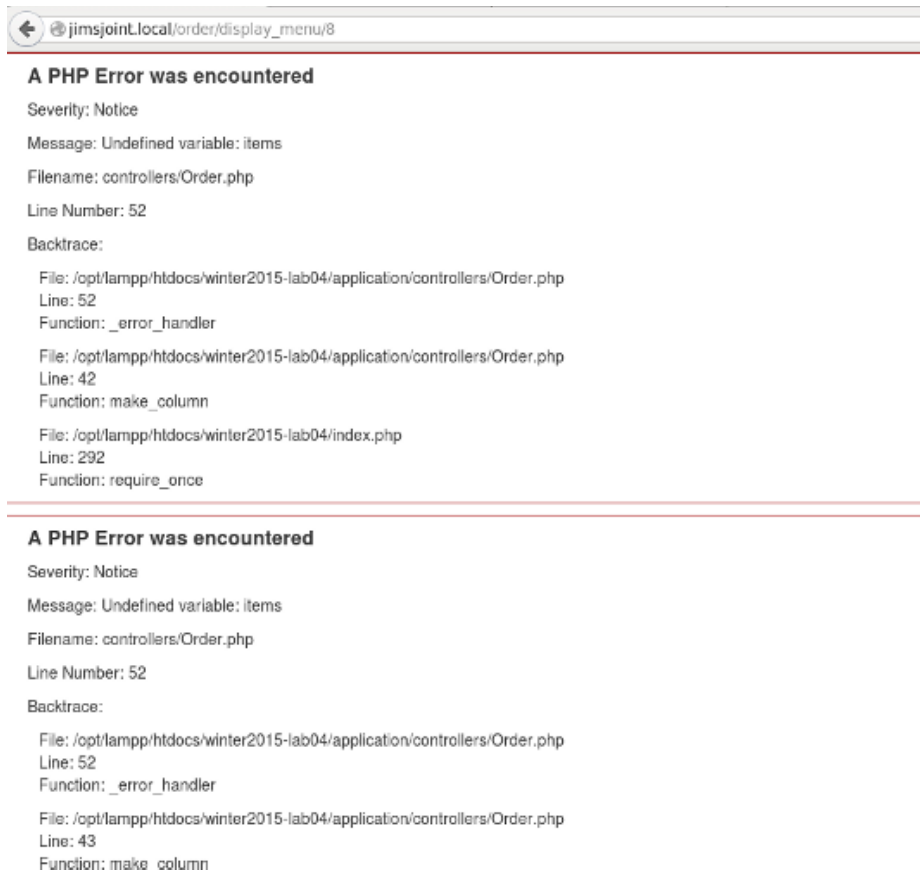
```
// start a new order
function neworder() {
    $order_num = $this->orders->highest() + 1;

    $neworder = $this->orders->create();
    $neworder->num = $order_num;
    $neworder->date = date();
    $neworder->status = 'a';
    $neworder->total = 0;
    $this->orders->add($neworder);

    redirect('/order/display_menu/' . $order_num);
}
```

## Post Step 3

That solved the redirection problem, but the order display (for our new order) has other issues - an undefined variable "items" :(



The screenshot shows a web browser window with the address bar displaying `jimsjoinLlocal/order/display_menu/8`. Below the address bar, a red banner indicates "A PHP Error was encountered". The error details are as follows:

- Severity: Notice
- Message: Undefined variable: items
- Filename: controllers/Order.php
- Line Number: 52
- Backtrace:
  - File: /opt/lampp/htdocs/winter2015-lab04/application/controllers/Order.php
  - Line: 52
  - Function: \_error\_handler
  - File: /opt/lampp/htdocs/winter2015-lab04/application/controllers/Order.php
  - Line: 42
  - Function: make\_column
  - File: /opt/lampp/htdocs/winter2015-lab04/index.php
  - Line: 292
  - Function: require\_once

Below this error message, another identical error message is shown, indicating the same issue persists.

---

## 4. HANDLE ORDER DISPLAY

---

There are two pieces here - building the title for the page (order # and total), and building the graphical menu display.

Fix controllers/order:display\_menu(). It needs to get order data from somewhere. It also needs to get the column data from somewhere.

---

### Fix the Title

---

You are given the order # as a parameter. Retrieve the order record from the orders table, for instance `$order = $this->orders->get($order_num)`. A starting point for the title would then be `$order->num` or `$order_num`. This is the first part of the FIXME at line 32.

You might end up with something like...

```
$this->data['title'] = "Order #  
" . $order_num .  
' (' .  
number_format($this->orders->total($order_num),  
2) . ')';
```

```
// add to an order  
function display_menu($order_num = null) {  
    if ($order_num == null)  
        redirect('/order/neworder');  
  
    $this->data['pagebody'] = 'show_menu';  
    $this->data['order_num'] = $order_num;  
    //FIXME  
    $this->data['title'] = "Order # " . $order_num;  
  
    // Make the columns  
    $this->data['meals'] = $this->make_column('m');  
    $this->data['drinks'] = $this->make_column('d');
```

# Order Display - Column Data

Inspecting views/show\_menu, the data for each column comes from an array of Menu records. The make\_column method in controllers/Order looks ready for that, and it is even passed the value of the category to use for relevant menu items.

```
// make a menu ordering column
function make_column($category) {
    return $this->menu->some('category',$category);
}

// add an item to an order
```

Sounds like the body of that method (FIXME on line 44) will be very similar to the fix we applied to the Welcome controller... return \$this->menu->some('category',\$category)

## Now We can See

Yay - we can now see the ordering display :)



### Order # 1

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)

# Order Display - Order Total

Fix `models/Orders::total()`. It needs to calculate the current total of an order. It will need to get all of the items that make up that order.

The last part of the menu display is showing the order total in the title. The `models/Orders::total()` method (lines 20+) is meant to handle that. In order to calculate the total for an order, iterate over the items in an order. For each, retrieve its corresponding menu item. Add the orderitem quantity times the menu price to the order total.

```
// calculate the total for an order
function total($num)
{
    $CI = & get_instance();
    $items = $CI->orderitems->group($num);
    $result = 0;
    if (count($items) > 0)
        foreach ($items as $item)
        {
            $menu = $CI->menu->get($item->item);
            $result += $item->quantity * $menu->price;
        }
    return $result;
}
```

# Menu Display Looking Good :)

Our menu display looks better now, with the current order total displayed as part of the page header...



**Order # 1 (0.00)**

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



Turkey (5.95)



Bubbly (14.50)



Cheese (2.95)



---

## Order Model Needs Updating Too

---

The order total is also a property in the orders table. The writeup says it only has to be saved/accurate when an order is completed. You could make a design decision to update the order total whenever an item was added to an order, in which case this step could be satisfied by getting that property from the order. Regardless of your design decision, you will need to complete the `Order::total()` method at some point.

However you calculate or get the order total, it needs to be appended to the page title, inside parentheses. It would be a good idea to format it nicely and consistently. Any of the `money_format`, `number_format` or `sprintf` functions, built-in to PHP, could do the trick.

---

## 5. HANDLE ORDER ADDITIONS

---

Fix `controllers/Order:add()`. It will need to use `models/Orders::add_item()` to do useful stuff.

This method is invoked when a menu picture is clicked on. You can see that by mousing over any of the menu item images and observing the target link in the bottom left of your browser window.

If all goes well, clicking on an item will add it to the order and redisplay the menu. The only evidence of this will be that the order total is updated.

At this point, with no logic behind the handling method, clicking on an item appears to do nothing, which is quite correct!

---

## Let's Start to Fix the Handling

---

This method is invoked when a menu picture is clicked on. You can see that by mousing over any of the menu item images and observing the target link in the bottom left of your browser window.

`Order::add` needs to properly update the `orderitems` table to reflect the request. In theory, the logic to do this could be in a controller (interpreting it as business logic), or in a model (interpreting it as model logic). Given the "model adapter" strategy that PHP frameworks favor, it makes most sense to deal with the logic inside the model, reducing the coupling between the controller and model layers.

---

## Order Additions - Facade

---

`Order::add(num,item)` then becomes just a facade for calling the related model method, `models/Orders::add_item(...)`. These have similar names to reinforce the connection between them.

So, the controller fix (line 50 fixme) is simply  
`$this->orders->add_item($order_num,$`

```
// calculate the total for an order
function total($num)
{
    $CI = & get_instance();
    $items = $CI->orderitems->group($num);
    $result = 0;
    if (count($items) > 0)
        foreach ($items as $item)
        {
            $menu = $CI->menu->get($item->item);
            $result += $item->quantity * $menu->price;
        }
    return $result;
}
```

---

## Order Additions - Orderitems?

---

The `add_item` method in the orders model now needs to be completed. There, you need to see if that item is part of the order already, in which case retrieve its record, increment the quantity and then update the table. If it isn't already there, you will need to make an empty orderitem record, and populate its fields appropriately (including setting the quantity to 1), before adding that record to the table.

The `Orderitems` model is not referenced inside any controller. I did this on purpose, hoping that you would conclude that such references belong in the `Orders` model. In the `Orders` model, you will need to get a handle to the CodeIgniter instance, in order to reference the `Orderitems` model, because of O-O scoping. Even if autoloaded, `Orderitems` would be a property of the controller object, and out of scope inside a model object.

## Order Additions - Orderitems!

The idea, inside models/Orders, is shown to the right.

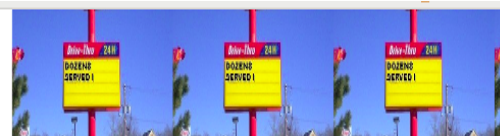
If you made the earlier design decision to keep an order's total continuously updated, then you would have that method here. Otherwise, you are done.

```
// add an item to an order
function add_item($num, $code)
{
    $CI = & get_instance();
    if ($CI->orderitems->exists($num, $code))
    {
        $record = $CI->orderitems->get($num, $code);
        $record->quantity++;
        $CI->orderitems->update($record);
    } else
    {
        $record = $CI->orderitems->create();
        $record->order = $num;
        $record->item = $code;
        $record->quantity = 1;
        $CI->orderitems->add($record);
    }
}
```

## Order Additions - Ideal Treatment?

Order additions should now work, with the order total updating as you click on items to add to an order.

An ideal implementation would have some domain/entity classes, specifically a domain/Order to deal with the logic here that might feel awkward in a controller or in a conventional orders model, which is part of the data access layer and not an entity encapsulation.



Order # 1 (8.90)

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



## 6. HANDLE CHECKOUT

Fix

`controllers/Order::checkout()`.  
Where do the order details come from?

`controllers/order::check_out()` displays the `show_order` view. You need to pass view parameters to this, or we will get a funny looking checkout screen, shown right.



## Checkout View Parameters

`order_num` and `total` come from the order object. You know how to get this.

Here is an implementation for the 'total' and 'items' view parameters:

```
// checkout
function checkout($order_num)
{
    $this->data['title'] = 'Checking Out';
    $this->data['pagebody'] = 'show_order';
    $this->data['order_num'] = $order_num;

    $this->data['total'] = number_format($this->orders->total($order_num), 2);

    $items = $this->orderitems->group($order_num);
    foreach ($items as $item)
    {
        $menuitem = $this->menu->get($item->item);
        $item->code = $menuitem->name;
    }
    $this->data['items'] = $items;

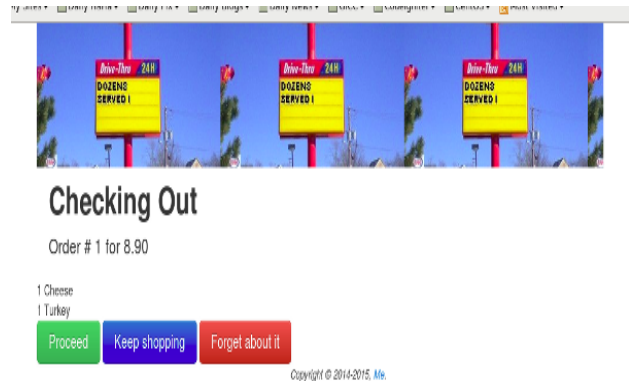
    $this->render();
}
```



# Checkout Is Starting to Look Real

Checkout is looking better (see right).

The intent is that the "Proceed" button be enabled only if the order is valid. We'll fix that next.



## Checkout - Order Validation

Validate the order before displaying the checkout page. To be considered "valid", it must include at least one item from each menu category.

models/Orders has a validate() method. Implement this, and then the view parameter becomes easy ...

```
$this->data['okornot'] =
$this->orders->validate($num);
```

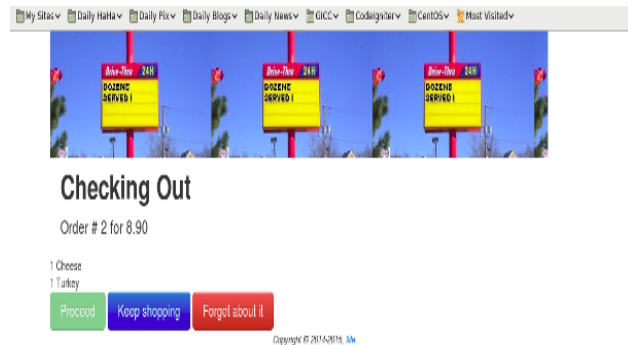
```
// validate an order
// it must have at least one item from each category
function validate($num)
{
    $CI = &get_instance();
    $items = $CI->orderitems->group($num);
    $gotem = array();
    if (count($items) > 0)
        foreach ($items as $item)
        {
            $menu = $CI->menu->get($item->item);
            $gotem[$menu->category] = 1;
        }
    return isset($gotem['m']) && isset($gotem['d']) && isset($gotem['s']);
}
```

# Checkout - Using Validation

We can reference `Orders::validate` from our `checkout()` method ...

```
$this->data['okornot'] =  
$this->orders->validate($num);
```

The checkout view should now show the Proceed button disabled.



## 7. HANDLE COMPLETION

Fix `controllers/Order::commit()` ?  
This will only be invocable if the Proceed button is enabled, i.e. the order is valid.

`controllers/Order::commit()` needs to update the order status, to 'c', and to make sure the date/time and the total are properly set too.

```
// proceed with checkout
function commit($order_num)
{
    if (!$this->orders->validate($order_num))
        redirect('/order/display_menu/' . $order_num);
    $record = $this->orders->get($order_num);
    $record->date = date(DATE_ATOM);
    $record->status = 'c';
    $record->total = $this->orders->total($order_num);
    $this->orders->update($record);
    redirect('/');
}
```

# Completion, Really

controllers/Order::cancel() needs to update the order status to 'x' (for cancelled), after deleting any related orderitems records. To maintain cohesion, there is a flush(num) method in the models/Orders class, for you to implement.

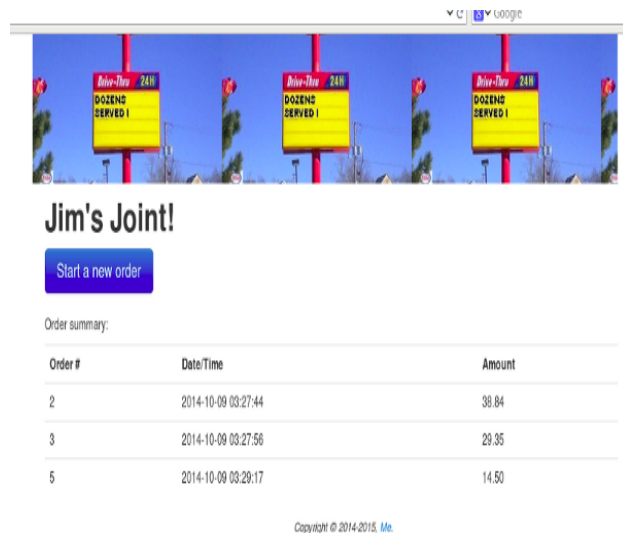
```
// cancel the order
function cancel($order_num)
{
    $this->orderitems->delete_some($order_num);
    $record = $this->orders->get($order_num);
    $record->status = 'x';
    $this->orders->update($record);
    redirect('/');
}
```

It should also retrieve the orderitems for an order, and then delete them. That would be done by the flush() method inside our Orders model, but I think we have done enough!

# Are We Done Yet?

Make sure the homepage displays completed orders, with correct totals!

This is more of a quality control step, making sure that the expected completed orders are shown :)



Jim's Joint!

Start a new order

Order summary:

Order #	Date/Time	Amount
2	2014-10-09 03:27:44	38.84
3	2014-10-09 03:27:56	29.95
5	2014-10-09 03:29:17	14.50

Copyright © 2014-2015, Ma

# Hint!

I warned you to skim the whole slideshow before beginning!

Your controllers only ever need to deal with the menu and orders models. Your orders model is the only place you need to deal with the orderitems model.

---

# Congratulations!

---

You have completed tutorial #ci-normal02: Working With CodeIgniter Models

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: lab05 Models

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.