INFS3202/7202 – Web Information Systems

Lecture Week 5:
MVC – Creating CRUD Applications

Dr Aneesha Bakharia (Senior Lecturer, EECS)
Email: a.bakharia1@uq.edu.au

Contents

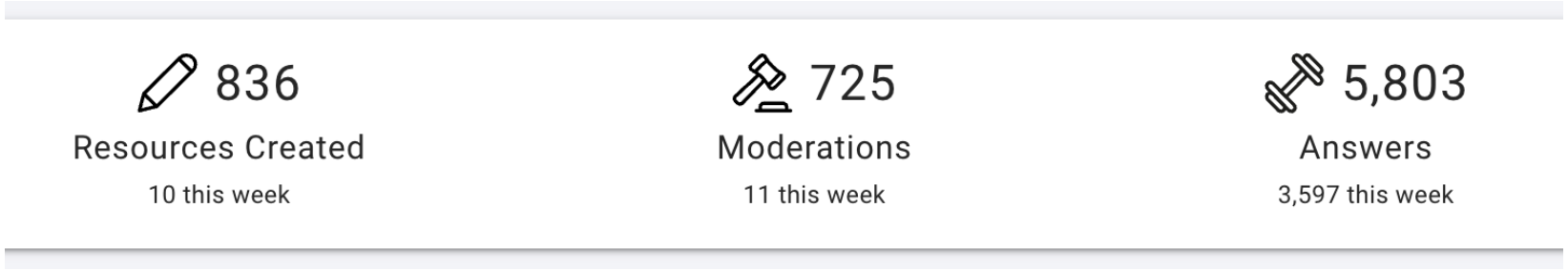
Course Updates

Issue/Feedback	Change
Week 4 Lab 3 was challenging	• It was designed to be challenging and introduce practical database design concepts (to help you with the Design Document Assessment Item and understand table relationships) and to see the full developer workflow (create tables, add data and then build a page to display the data) • If you can complete Lab 3, it sets the basis for course success • Labs are not graded and unlike last year there are no in Lab graded quizzes – so you can learn at your pace and use in your project which is due at the end of Week 12
Cover more on Nginx	• Will cover in Week 8
Will real world company architectures be covered	• Week 10 and 12. In Week 10 AWS Solution Architects will give a guest lecture and you'll learn to setup your own server from Scratch in the Lab.
Design Document Assessment Item	• Due Monday of Week 7

RiPPLE Weekly Assessment Item
End Cycle 1

RiPPLE Cycle 1

- The first RiPPLE (Create, Moderate & Practice) cycle has ended
- RiPPLE activities are designed to trigger higher level thinking by allowing you to research a topic, create questions with answers, review other students questions and finally complete/practice questions
- Very impressed by the quality of questions, moderations and the number of questions students have been answering.



- A change in Rating does not affect your grade – it just means that if your Rating goes negative for a question that RiPPLE will give you more questions in the same area to help you improve
- New RiPPLE cycle 2 is now open – In Week 5 you need to create 3 questions on any of these topics: Databases, SQL, CodeIgniter Models and CodeIgniter Migrations

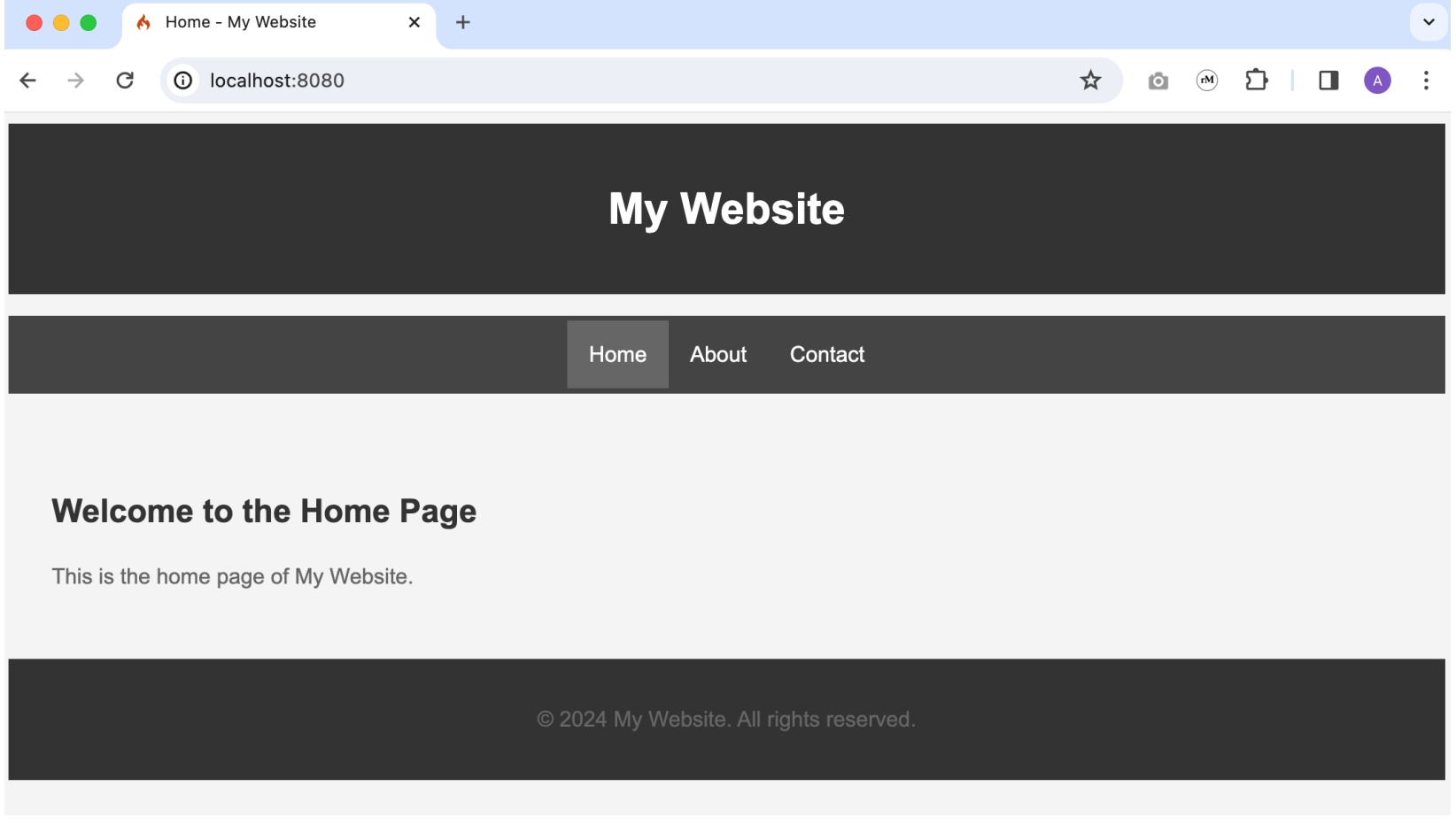
Server-side Web Dev Patterns Covered in INFS3202/7202 so far

Base Template View (Lecture 3 and Lab 2)

- Create a HTML template that keeps the full page HTML (including the Header and Footer)
- Define sections for content
  - Can have multiple sections
- Other views can inherit the template and then just need to include content for the sections

- Logic in the Nav bar can be included to highlight the currently view page

- Advantage
  - Reduces code duplication
  - Keeps HTML in base template complete (Making it easier to edit the HTML)



Creating a Base View Template

Create section using renderSection() for each section that will change

There can be multiple named sections

app > Views > base_template.php

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Home - My Website</title>
  <link rel="stylesheet" href="<?= base_url('styles/style.css'); ?>">

</head>
<body>
  <header>
    <h1>My Website</h1>
  </header>

  <nav>
    <ul>
        <li><a href="/" class="<?= service('request')->getUri()->getPath() == '' ? 'active' : ''; ?>">Home</a></li>
        <li><a href="/about" class="<?= service('request')->getUri()->getPath() == 'about' ? 'active' : ''; ?>">About</a></li>
        <li><a href="/contact" class="<?= service('request')->getUri()->getPath() == 'contact' ? 'active' : ''; ?>">Contact</a></li>
    </ul>
  </nav>

  <main>
    <?= $this->renderSection('content') ?> <!-- Placeholder for page content -->
  </main>

  <footer>
    <p>&copy; &copy; <?= date('Y') ?> My Website. All rights reserved.</p>
  </footer>
</body>
</html>


```

Using a Base template

home.php

app > Views > home.php

```php
<?= $this->extend('base_template') ?>
<?= $this->section('content') ?>

<h2>Welcome to the Home Page</h2>
<p>This is the home page of My Website.</p>

<?= $this->endSection() ?>
```

about.php

app > Views > about.php

```php
<?= $this->extend('base_template') ?>
<?= $this->section('content') ?>

<h2>About Us</h2>
<p>We are a passionate team working on amazing projects.</p>

<?= $this->endSection() ?>
```

contact.php

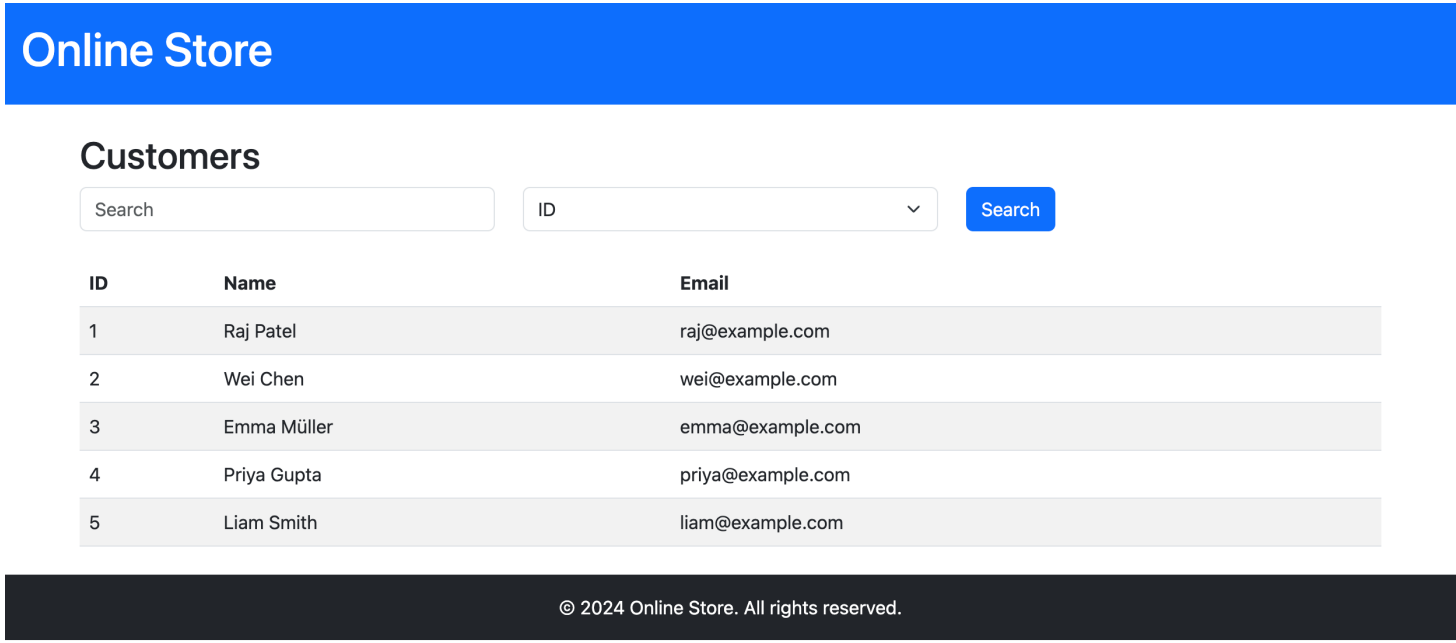
app > Views > contact.php

```php
<?= $this->extend('base_template') ?>
<?= $this->section('content') ?>

<h2>Contact Us</h2>
<p>Feel free to get in touch with us.</p>

<?= $this->endSection() ?>
```

Search and List Multiple Records
Week 4 – Customers Example

- Created:
  - a Database
  - Tables (using a Migration)
  - Model
  - Routes
  - Controller

- Retrieves all records from the customers table
- Displays it in a table



- Also allows searching on fields
- Is able to filter records and only return what matches the search criteria

Search and List Multiple Records
Week 4 – Customers Example - Routes

- Two routes
  - /
  - /customers

cidemo > app > Config > Routes.php

```php
<?php

use CodeIgniter\Router\RouteCollection;

/**
 * @var RouteCollection $routes
 */
//$routes->get('/', 'Home::index');

$routes->get('/', 'CustomerController::index');
$routes->get('/customers', 'CustomerController::index');

```

- Routes are in the app/Config/Routes.php file

Search and List Multiple Records
Week 4 – Customers Example - Model

- A model is a class that maps to a table in your database

- It allows us to use object notation to:
  - Filter and search records in a table
  - create new records
  - update a record
  - delete records

cidemo > app > Models > Customers.php

```php
<?php

namespace App\Models;

use CodeIgniter\Model;

class Customers extends Model
{
    protected $table            = 'customers';
    protected $primaryKey       = 'id';
    protected $useAutoIncrement = true;
    protected $returnType       = 'array';
    protected $useSoftDeletes   = false;
    protected $protectFields    = true;
    protected $allowedFields    = [];

}
```

- https://codeigniter4.github.io/CodeIgniter4/dbmgmt/forge.html
- Models are placed in the app/Models folder

Search and List Multiple Records
Week 4 – Customers Example - Controller

- The controller can load a model
- Get form field values
- Use logic to decide to filter the records or retrieve all records
- Pass returned data to the view

- Controllers are placed in the app/Controllers folder

cidemo > app > Controllers > CustomerController.php

```php
<?php

namespace App\Controllers;

use App\Models\CustomerModel;

class CustomerController extends BaseController
{

    public function __construct()
    {
        // Load the URL helper, it will be useful in the next steps
        // Adding this within the __construct() function will make it
        // available to all views in the ResumeController
        helper('url');
    }

    public function index()
    {
        $model = new \App\Models\Customers();

        $searchField = $this->request->getGet('search_field');
        $searchValue = $this->request->getGet('search_value');

        if (!empty($searchField) && !empty($searchValue)) {
            $customers = $model->where($searchField, $searchValue)->findAll();
        } else {
            $customers = $model->findAll();
        }

        $data = [
            'customers' => $customers,
            'searchField' => $searchField,
            'searchValue' => $searchValue,
        ];

        return view('customers', $data);
    }
}

```

Search and List Multiple Records
Week 4 – Customers Example - View

- Inherits from the base template
  - We don’t have duplicated header and footer HTML to worry about
  - Wrap the content in a named section

cidemo > app > Views > customers.php

```php
<?= $this->extend('base_template') ?>

<?= $this->section('content') ?>
<h2>Customers</h2>
```

- We can create a HTML table
  - Columns for each field
  - Loop over the $customers data

```php
<table class="table table-striped">
    <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Email</th>
        </tr>
    </thead>
    <tbody>
        <?php foreach ($customers as $customer): ?>
            <tr>
                <td><?= $customer['id'] ?></td>
                <td><?= $customer['name'] ?></td>
                <td><?= $customer['email'] ?></td>
            </tr>
        <?php endforeach; ?>
    </tbody>
</table>
<?= $this->endSection() ?>
```

Search and List Multiple Records
Week 4 – Customers Example – View with Form

- Views can have HTML forms
- Forms can collect data
- In this example we have a:
  - Text input
  - Drop down
- The form is a search form and we can select which field in the database table we want to search
- Clicking on the Submit button sends the form data to the server
- The route will be sent to the matching Controller

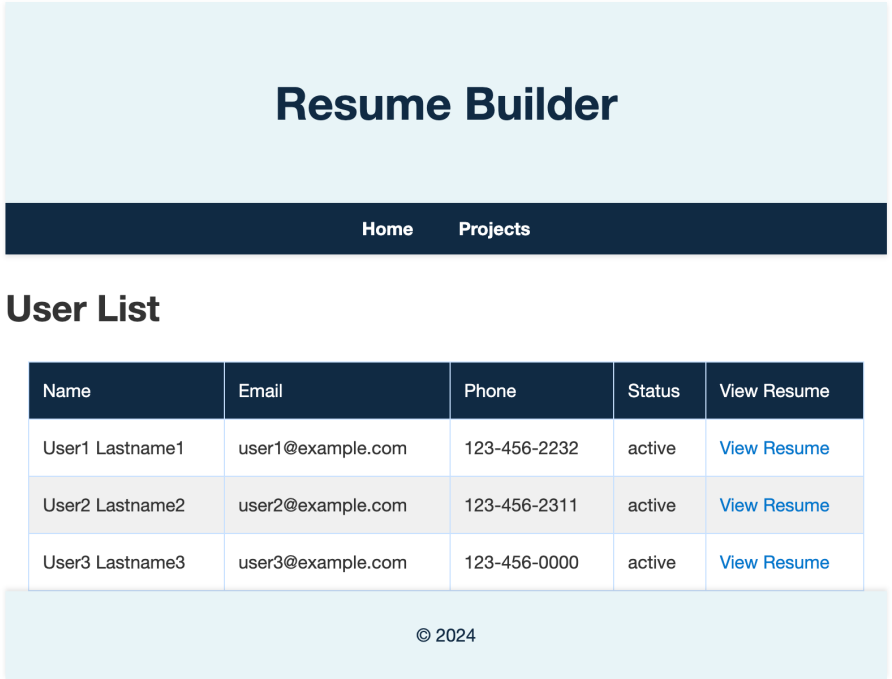
cidemo > app > Views > customers.php

```php
<?= $this->extend('base_template') ?>

<?= $this->section('content') ?>
<h2>Customers</h2>

<form method="get" action="<?= base_url('customers'); ?>" class="mb-4">
    <div class="row">
        <div class="col-md-4">
            <input type="text" name="search_value" class="form-control" placeholder="Search" value="<?= $searchValue ?>">
        </div>
        <div class="col-md-4">
            <select name="search_field" class="form-select">
                <option value="id" <?= $searchField === 'id' ? 'selected' : '' ?>>ID</option>
                <option value="name" <?= $searchField === 'name' ? 'selected' : '' ?>>Name</option>
                <option value="email" <?= $searchField === 'email' ? 'selected' : '' ?>>Email</option>
            </select>
        </div>
        <div class="col-md-4">
            <button type="submit" class="btn btn-primary">Search</button>
        </div>
    </div>
</form>

```

List - Detail Page
Week 4 Lab 3 – Resume Builder

- Created:
  - a Database
  - Tables (using a Migration)
  - Models
  - Routes
  - Controller

- Displays a table with all Users
- Has a link to each Resume
- Essentially an Admin panel for the SaaS



List Page
Week 4 Lab 3 – Resume Builder – Migration

- A migration was used to create the Users table
- A migration allows us to version database tables
- We need to define the fields using CodeIgniter Database Forge syntax
- https://codeigniter4.github.io/CodeIgniter4/dbmgmt/forge.html

app > Database > Migrations > 2024-03-12-013743_CreateUsersTable.php

```php
<?php

namespace App\Database\Migrations;

use CodeIgniter\Database\Migration;

class CreateUserTable extends Migration
{
    public function up()
    {
        // Define the User table
        $this->forge->addField([
            'user_id' => [
                'type' => 'INT',
                'constraint' => 11,
                'unsigned' => TRUE,
                'auto_increment' => TRUE
            ],
            'name' => [
                'type' => 'VARCHAR',
                'constraint' => '255',
            ],
            'email' => [
                'type' => 'VARCHAR',
                'constraint' => '255',
            ],
            'phone' => [
                'type' => 'VARCHAR',
                'constraint' => '20',
            ],
            'url' => [
                'type' => 'VARCHAR',
                'constraint' => '255',
            ],
            'summary' => [
                'type' => 'TEXT',
            ],
        ]);

        $this->forge->addKey('user_id', TRUE); // Set user_id as primary key
        $this->forge->createTable('User'); // Create the User table
    }

```

List Page
Week 4 Lab 3 – Resume Builder – Model

- A model is a class that maps to a table in your database

- It allows us to use object notation to:
  - Filter and search records in a table
  - create new records
  - update a record
  - delete records

app > Models > UserModel.php

```php
<?php

namespace App\Models; // Declares the namespace for this file which helps in organizing and grouping the classes.

use CodeIgniter\Model; // Imports the base Model class from the CodeIgniter framework.

class UserModel extends Model // Defines a new class UserModel that extends CodeIgniter's Model class.
{
    protected $table = 'User'; // Specifies the database table that this model should interact with.
    protected $primaryKey = 'user_id'; // Defines the primary key field of the table for CRUD operations.
    // Lists the fields that are allowed to be set using the model. This is for security and prevents mass assignment vulnerabilities.
    protected $allowedFields = ['name', 'email', 'phone', 'url', 'summary', 'status'];
    protected $returnType = 'array'; // Sets the default return type of the results. This model will return results as arrays.
}

```

- Models are placed in the app/Models folder

List Page
Week 4 Lab 3 – Resume Builder – Controller

- The controller can load a model
- Get form field values
- Use logic to decide to filter the records or retrieve all records
- Pass returned data to the view
- Controllers are placed in the app/Controllers folder

app > Controllers > ResumeController.php

```php
<?php namespace App\Controllers;

class ResumeController extends BaseController
{

    public function __construct()
    {
        // Load the URL helper, it will be useful in the next steps
        // Adding this within the __construct() function will make it
        // available to all views in the ResumeController
        helper('url');
    }

    public function admin()
    {
        // Creates an instance of the UserModel class.
        // It's important to use the fully qualified namespace to ensure the correct class is instantiated.
        $model = new \App\Models\UserModel();

        // Prepares to fetch data from the database using the UserModel.
        // The data fetched will be stored in the array `$data` under the key 'users'.
        $data['users'] = $model->orderBy('name', 'ASC') // This method call specifies the order of the results.
                               // The 'name' column is used to sort the users in ascending order (ASC).
                               // If the sorting needs to be done by 'surname', replace 'name' with 'surname'.
                               ->where('status', 'active') // This adds a condition to the query.
                               // Only users with their 'status' column set to 'active' will be included in the results.
                               ->findAll(); // Executes the query with the specified conditions and ordering.
                               // The `findAll` method fetches all records that match the query conditions.
                               // The results are returned and assigned to `$data['users']`.

        return view('admin', $data);
    }
```

Detail Page
Week 4 Lab 3 – Resume Builder – Models

app > Models > EducationModel.php

```php
<?php

namespace App\Models;

use CodeIgniter\Model;

class EducationModel extends Model
{
    protected $table = 'Education'; // Set to your actual table name
    protected $primaryKey = 'education_id';
    protected $allowedFields = ['user_id', 'institution', 'url', 'area', 'studyType',
    protected $returnType = 'array';
    protected $useTimestamps = false; // Adjust based on your table's design
}

```

app > Models > WorkModel.php

```php
<?php

namespace App\Models;

use CodeIgniter\Model;

class WorkModel extends Model
{
    protected $table = 'Work'; // Set to your actual table name
    protected $primaryKey = 'work_id';
    protected $allowedFields = ['user_id', 'name', 'position', 'url', 'startDate', 'endDate', 'summary'];
    protected $returnType = 'array';
    protected $useTimestamps = false; // Modify as needed
}

```

app > Models > AddressModel.php

```php
<?php

namespace App\Models;

use CodeIgniter\Model;

class AddressModel extends Model
{
    protected $table = 'Address'; // Set to your actual table name
    protected $primaryKey = 'address_id';
    protected $allowedFields = ['user_id', 'address', 'postalCode', 'city',
    protected $returnType = 'array';
    protected $useTimestamps = false; // Set to true if you have created_at
}

```

- 3 Models
- All relate to the Users table

Detail Page
Week 4 Lab 3 – Resume Builder – Routes

- Defined the /resume route
- Accepts a numeric id
  - We use this id to retrieve the resume

app > Config > Routes.php

```php
<?php

use CodeIgniter\Router\RouteCollection;

/**
 * @var RouteCollection $routes
 */
//$routes->get('/', 'Home::index');

$routes->get('/', 'ResumeController::admin');

$routes->get('/admin', 'ResumeController::admin');

// This route pattern matches URLs like /resume/1, where 1 is the user_id. The (:num) segment
// ensures that only numeric IDs are matched, and $1 passes the captured ID to the resume method in the ResumeController.
$routes->get('/resume/(:num)', 'ResumeController::resume/$1');

```

Detail Page
Week 4 Lab 3 – Resume Builder – Routes

- Defined the /resume route
- Accepts a numeric id
  - We use this id to retrieve the resume

app > Config > Routes.php

```php
<?php

use CodeIgniter\Router\RouteCollection;

/**
 * @var RouteCollection $routes
 */
//$routes->get('/', 'Home::index');

$routes->get('/', 'ResumeController::admin');

$routes->get('/admin', 'ResumeController::admin');

// This route pattern matches URLs like /resume/1, where 1 is the user_id. The (:num) segment
// ensures that only numeric IDs are matched, and $1 passes the captured ID to the resume method in the ResumeController.
$routes->get('/resume/(:num)', 'ResumeController::resume/$1');

```

Detail Page
Week 4 Lab 3 – Resume Builder – Controller

- The route passes an id to the controller method
- Multiple models can be loaded

```php
public function resume($user_id)
{
    $userModel = new \App\Models\UserModel();
    $addressModel = new \App\Models\AddressModel();
    $educationModel = new \App\Models\EducationModel();
    $workModel = new \App\Models\WorkModel();

    // Fetch user details by user_id
    $data['user'] = $userModel->find($user_id);

    // Ensure user exists
    if (!$data['user']) {
        throw new \CodeIgniter\Exceptions\PageNotFoundException('User Not Found');
    }

    // Fetch related data
    $data['address'] = $addressModel->where('user_id', $user_id)->first();
    $data['educations'] = $educationModel->where('user_id', $user_id)->findAll();
    $data['works'] = $workModel->where('user_id', $user_id)->findAll();

    return view('resume_view', $data);
}
```

- We can get the user data, and the related address, education and work records
- All Resume data can be passed to the view

Detail Page
Week 4 Lab 3 – Resume Builder – View

- We can display a full Resume made up of data from multiple related tables

app > Views > resume_view.php

```php
<?= $this->extend('template') ?>
<?= $this->section('content') ?>

<h1>Resume of <?= esc($user['name']) ?></h1>
    <h2>Personal Information</h2>
    <p>Email: <?= esc($user['email']) ?></p>
    <p>Phone: <?= esc($user['phone']) ?></p>
    <p>Website: <a href="<?= esc($user['url'], 'attr') ?>"><?= esc($user['url']) ?></a></p>

    <?php if ($address): ?>
    <h2>Address</h2>
    <p><?= esc($address['address']) ?><br>
       <?= esc($address['city']) ?>, <?= esc($address['state']) ?><br>
       <?= esc($address['postalCode']) ?>, <?= esc($address['countryCode']) ?></p>
    <?php endif; ?>

    <h2>Education</h2>
    <table>
        <tr><th>Institution</th><th>Degree</th><th>Field of Study</th><th>Years</th></tr>
        <?php foreach ($educations as $education): ?>
            <tr>
                <td><?= esc($education['institution']) ?></td>
                <td><?= esc($education['studyType']) ?></td>
                <td><?= esc($education['area']) ?></td>
                <td><?= esc($education['startDate']) ?> - <?= esc($education['endDate']) ?></td>
            </tr>
        <?php endforeach; ?>
    </table>

    <h2>Work Experience</h2>
    <table>
        <tr><th>Company</th><th>Position</th><th>Period</th><th>Summary</th></tr>
        <?php foreach ($works as $work): ?>
            <tr>
                <td><?= esc($work['name']) ?></td>
                <td><?= esc($work['position']) ?></td>
                <td><?= esc($work['startDate']) ?> - <?= esc($work['endDate']) ?></td>
                <td><?= esc($work['summary']) ?></td>
                </tr>
        <?php endforeach; ?>
    </table>

    <p></p>

<?= $this->endSection() ?>
```

Create Read Update Delete (CRUD)

Code in week5_code/

CRUD

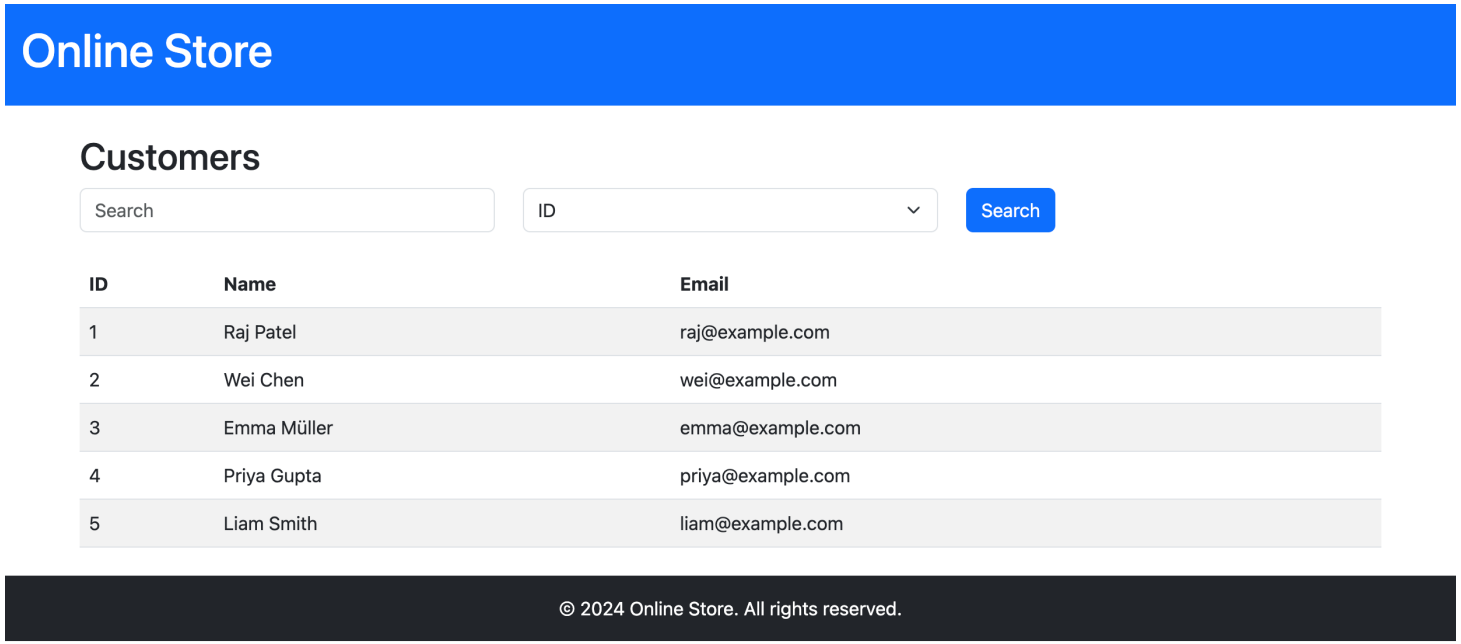
- **Definition of CRUD**:
  CRUD stands for Create, Read, Update, and Delete.
  These are the four basic functions of persistent storage.

- **Foundation of User Interfaces**:
  CRUD operations form the foundation of user interfaces in many applications, enabling users to interact directly with data.
- **Ubiquity in Web Applications**:
  Almost every web application has a backend database and uses CRUD operations to interact with the data.

- **APIs and CRUD**:
  Many RESTful APIs are designed around CRUD operations, mapping HTTP methods (POST, GET, PUT/PATCH, DELETE) to CRUD functionalities.
- **Database Operations**:
  At the database level, CRUD corresponds to SQL commands INSERT, SELECT, UPDATE, and DELETE, respectively.

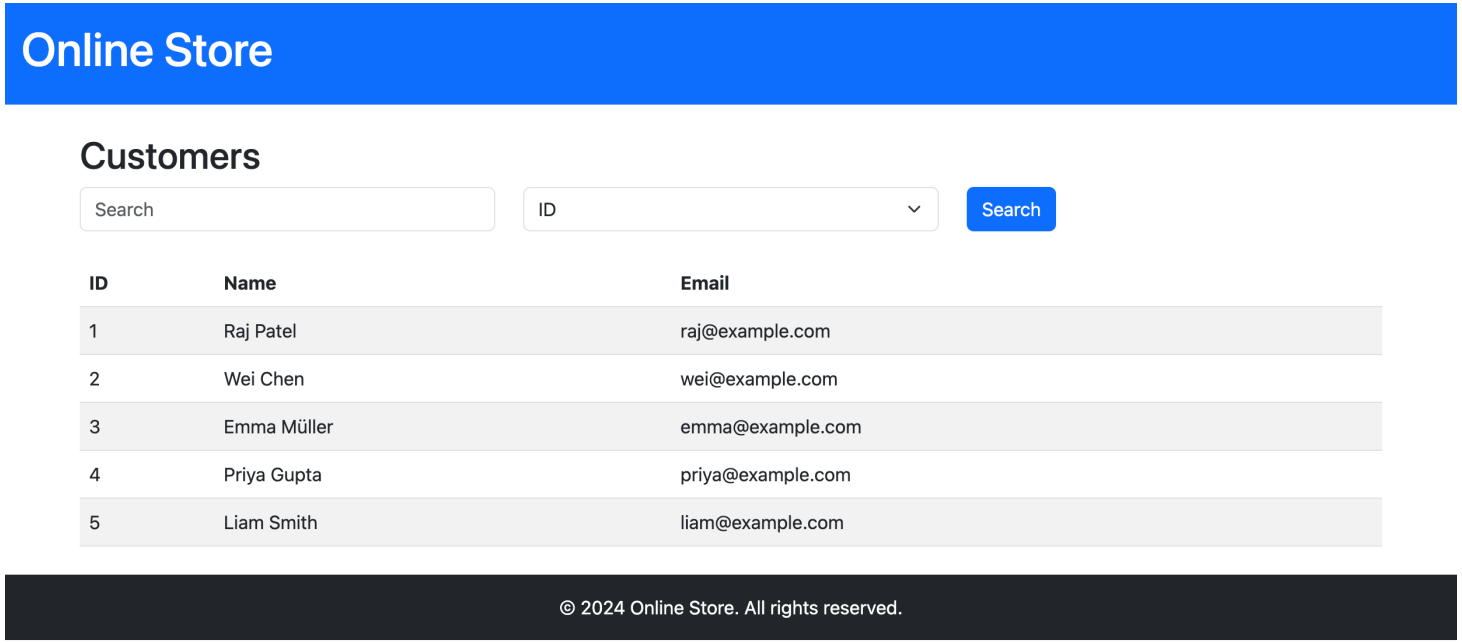
Adding CRUD to the Customers App

- **What do we need to add to this UI to make a CRUD application?**



Adding CRUD to the Customers App

- **Need an**
  - Add button that displays an Add form
  - Edit button in each row
  - Delete button in each row
  - Display the status of the last operation



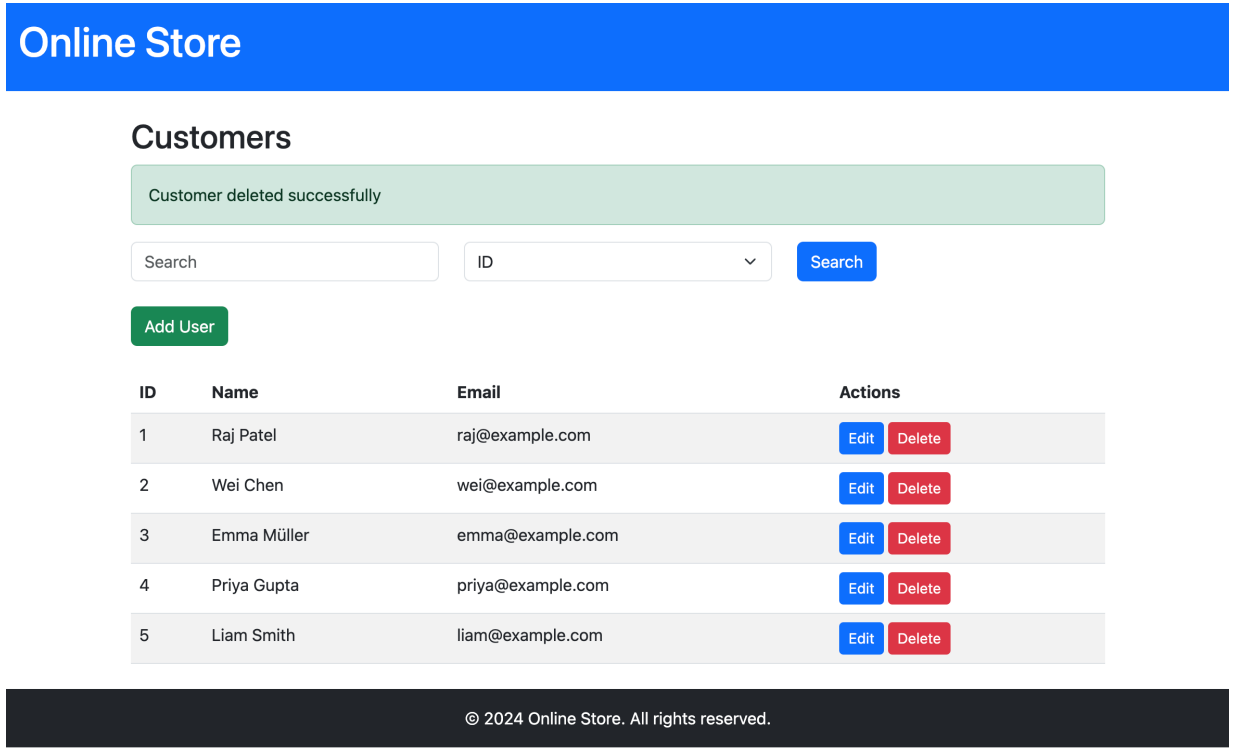
Adding CRUD to the Customers App

- We use the Bootstrap CSS Framework
- We have added the Button and Alert components
https://getbootstrap.com/docs/5.3/components/buttons/
&
https://getbootstrap.com/docs/5.3/components/alerts/

Primary Secondary Success Danger Warning Info Light Dark Link

HTML

```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-dark">Dark</button>

<button type="button" class="btn btn-link">Link</button>
```



A simple success alert—check it out!

HTML

```
<div class="alert alert-primary" role="alert">
  A simple primary alert—check it out!
</div>
<div class="alert alert-secondary" role="alert">
  A simple secondary alert—check it out!
</div>
<div class="alert alert-success" role="alert">
  A simple success alert—check it out!
</div>
<div class="alert alert-danger" role="alert">
  A simple danger alert—check it out!
</div>
<div class="alert alert-warning" role="alert">
  A simple warning alert—check it out!
</div>
```

Adding CRUD to the Customers App – View (pt 1)

- Added an Add button that links to customers/addform
- Edit & Delete buttons in each row

cidemo > app > Views > customers.php

```
1   <?= $this->extend('base_template') ?>
2
3   <?= $this->section('content') ?>
4   <h2>Customers</h2>
5
6   <?php if (session()->getFlashdata('success')): ?>
7       <div class="alert alert-success"><?= session()->getFlashdata('success') ?></div>
8   <?php endif; ?>
9
10  <?php if (session()->getFlashdata('error')): ?>
11      <div class="alert alert-danger"><?= session()->getFlashdata('error') ?></div>
12  <?php endif; ?>
13
14  <form method="get" action="<?= base_url('customers'); ?>" class="mb-4">…
30  </form>
31
32  <div class="mb-4">
33      <a href="<?= base_url('customers/addform') ?>" class="btn btn-success">Add User</a>
34  </div>
35
36  <table class="table table-striped">
37      <thead>…
44      </thead>
45      <tbody>
46          <?php foreach ($customers as $customer): ?>
47              <tr>
48                  <td><?= $customer['id'] ?></td>
49                  <td><?= $customer['name'] ?></td>
50                  <td><?= $customer['email'] ?></td>
51                  <td>
52                      <a href="<?= base_url('customers/editform/' . $customer['id']) ?>" class="btn btn-primary btn-sm">Edit</a>
53                      <a href="<?= base_url('customers/delete/' . $customer['id']) ?>" class="btn btn-danger btn-sm"
54                          onclick="return confirm('Are you sure you want to delete this customer?')">Delete</a>
55                  </td>
56              </tr>
57          <?php endforeach; ?>
58      </tbody>
59  </table>
60  <?= $this->endSection() ?>
```

Adding CRUD to the Customers App – View (pt 2)

- Within a session we can store flash data
- We store a success or error messages
- We can display an alert with the message

cidemo > app > Views > customers.php

```
<?= $this->extend('base_template') ?>

<?= $this->section('content') ?>
<h2>Customers</h2>

<?php if (session()->getFlashdata('success')): ?>
    <div class="alert alert-success"><?= session()->getFlashdata('success') ?></div>
<?php endif; ?>

<?php if (session()->getFlashdata('error')): ?>
    <div class="alert alert-danger"><?= session()->getFlashdata('error') ?></div>
<?php endif; ?>

<form method="get" action="<?= base_url('customers'); ?>" class="mb-4">…
</form>

<div class="mb-4">
    <a href="<?= base_url('customers/addform') ?>" class="btn btn-success">Add User</a>
</div>

<table class="table table-striped">
    <thead>…
    </thead>
    <tbody>
        <?php foreach ($customers as $customer): ?>
            <tr>
                <td><?= $customer['id'] ?></td>
                <td><?= $customer['name'] ?></td>
                <td><?= $customer['email'] ?></td>
                <td>
                    <a href="<?= base_url('customers/editform/' . $customer['id']) ?>" class="btn btn-primary btn-sm">Edit</a>
                    <a href="<?= base_url('customers/delete/' . $customer['id']) ?>" class="btn btn-danger btn-sm"
                                onclick="return confirm('Are you sure you want to delete this customer?')">Delete</a>
                </td>
            </tr>
        <?php endforeach; ?>
    </tbody>
</table>
<?= $this->endSection() ?>
```

Adding CRUD to the Customers App – Add/Edit Form

- One form for both the Add and Edit
- Logic to determine to save (new record) or update (existing customer with an id)
- Logic to change the title of the button – Update or Save

cidemo > app > Views > customer_form.php

```
1   <?= $this->extend('base_template') ?>
2
3   <?= $this->section('content') ?>
4   <h2><?= $title ?></h2>
5
6   <form method="post" action="<?= $customer ? base_url('customers/update/' . $customer['id']) : base_url('customers/save') ?>">
7       <div class="mb-3">
8           <label for="name" class="form-label">Name</label>
9           <input type="text" name="name" id="name" class="form-control" value="<?= $customer ? $customer['name'] : '' ?>" required>
10      </div>
11      <div class="mb-3">
12          <label for="email" class="form-label">Email</label>
13          <input type="email" name="email" id="email" class="form-control" value="<?= $customer ? $customer['email'] : '' ?>" required>
14      </div>
15      <button type="submit" class="btn btn-primary"><?= $customer ? 'Update' : 'Save' ?></button>
16  </form>
17  <?= $this->endSection() ?>
```

- The form method is set to post
- The required attribute will give us client-side validation

Adding CRUD to the Customers App - Routes

- We could reduce the routes and have a single controller that handles both the get and post (add and edit) but let's keep it simple this time.
- Note difference between get and post

cidemo > app > Config > Routes.php

```php
<?php

use CodeIgniter\Router\RouteCollection;

/**
 * @var RouteCollection $routes
 */
//$routes->get('/', 'Home::index');

$routes->get('/', 'CustomerController::index');
$routes->get('/customers', 'CustomerController::index');

// The Router to display an Add Customer Form
$routes->get('/customers/addform', 'CustomerController::addform');
// The Router when the Save (Submit) button is clicked on the Add Form
$routes->post('/customers/save', 'CustomerController::save');

// The Router to display an Edit Customer Form for an existing customer
$routes->get('/customers/editform/(:num)', 'CustomerController::editform/$1');
// The Router when the Save button is clicked on the Edit Form (for an existing customer)
$routes->post('/customers/update/(:num)', 'CustomerController::update/$1');


// The Router when the Delete button is clicked
$routes->get('/customers/delete/(:num)', 'CustomerController::delete/$1');

```

Adding CRUD to the Customers App – Controller (pt 1)

- When adding a new customer the customer_form view is displayed
- The data passed to the view is null

cidemo > app > Controllers > CustomerController.php

```
2
3   namespace App\Controllers;
4
5   use App\Models\CustomerModel;
6
7   class CustomerController extends BaseController
8   {
9
10      public function __construct()
11  >   {…
18      }
19
20      public function index()
21  >   {…
40      }
41
42      public function addform()
43      {
44          $data = [
45              'title' => 'Add Customer',
46              'customer' => null,
47          ];
48
49          return view('customer_form', $data);
50      }
```

Adding CRUD to the Customers App – Controller (pt 2)

- We load the session service for all methods in the controller
- In the save() method we get the posted fields
- We can then pass the data to the save method of the model
- Finally, we set the Flash data message
- Note: Using Models really reduces code!

cidemo > app > Controllers > CustomerController.php

```php
<?php
namespace App\Controllers;

use App\Models\CustomerModel;

class CustomerController extends BaseController
{

    public function __construct()
    {
        // Load the URL helper, it will be useful in the next steps
        // Adding this within the __construct() function will make it
        // available to all views in the ResumeController
        helper('url');

        $this->session = \Config\Services::session();
    }

    public function index()
    {…
    }

    public function addform()
    {…
    }

    public function save()
    {
        $model = new \App\Models\Customers();

        $data = [
            'name' => $this->request->getPost('name'),
            'email' => $this->request->getPost('email'),
        ];

        if ($model->save($data)) {
            $this->session->setFlashdata('success', 'Customer added successfully');
        } else {
            $this->session->setFlashdata('error', 'Failed to add customer');
        }

        return redirect()->to('customers');
    }
```

Adding CRUD to the Customers App – Controller (pt 3)

- The id of the record being edited is passed from the Route to the controller
- We can load the record using the find() method, passing in the id
- Pass the record to the form view
- Note: Using Models really reduces code!

cidemo > app > Controllers > CustomerController.php

```php
<?php
namespace App\Controllers;

use App\Models\CustomerModel;

class CustomerController extends BaseController
{

    public function __construct()
    {…
    }

    public function index()
    {…
    }

    public function addform()
    {…
    }

    public function save()
    {…
    }

    public function editform($id)
    {
        $model = new \App\Models\Customers();

        $customer = $model->find($id);

        if (!$customer) {
            throw new \CodeIgniter\Exceptions\PageNotFoundException('Customer not found');
        }

        $data = [
            'title' => 'Edit Customer',
            'customer' => $customer,
        ];

        return view('customer_form', $data);
    }
```

Adding CRUD to the Customers App – Controller (pt 4)

- The id of the record being deleted is passed from the Route to the controller
- We can delete the record using the delete() method, passing in the id
- We set the flash data message
- Note: Using Models really reduces code!

cidemo > app > Controllers > CustomerController.php

```
52  {…
67  }
68
69  public function editform($id)
70  {…
85  }
86
87  public function update($id)
88  {
89      $model = new \App\Models\Customers();
90
91      $data = [
92          'name' => $this->request->getPost('name'),
93          'email' => $this->request->getPost('email'),
94      ];
95
96      if ($model->update($id, $data)) {
97          $this->session->setFlashdata('success', 'Customer updated successfully');
98      } else {
99          $this->session->setFlashdata('error', 'Failed to update customer');
100     }
101
102     return redirect()->to('customers');
103 }
104
105 public function delete($id)
106 {
107     $model = new \App\Models\Customers();
108
109     if ($model->delete($id)) {
110         $this->session->setFlashdata('success', 'Customer deleted successfully');
111     } else {
112         $this->session->setFlashdata('error', 'Failed to delete customer');
113     }
114
115     return redirect()->to('customers');
116 }
117 }
```

Summary

- You now should know how to:
  - Use a database to store data for your Web Application
  - Create list – detail pages
  - Create a search – list of results page
  - Add, Edit and Delete from your database
  - **We can however still optimise by merging the Add/Edit Controllers, add server side form field validation and enable security to ensure the form is run from your server (i.e. not cross-site form posting)**
    (In the Week 6 Lecture, you'll learn how to do these things)
- This forms the foundation of all Web Applications
  - What you can build now just depends on your imagination?

Once you learn to login and authenticate users and some Javascript

Designing a Database

(a step-by-step walkthrough to assist with the Design Document Assessment Item)

Database Design Goals

- Each table must represent a single subject
- No data item will be unnecessarily stored in more than one table, i.e., No data redundancy
- All non-key attributes in a table are dependent on the primary key
- Each table is void of insertion, update, deletion anomalies
- Objective of normalisation is to ensure that all tables are in at least 3rd Normal Form (3NF)

Let’s look at an example

StudentID	StudentName	CourseID	CourseName	CourseCredits	InstructorName
1	John Doe	CS101	Programming	3	Dr. Smith
1	John Doe	MAT201	Calculus	4	Prof. Johnson
2	Jane Smith	CS101	Programming	3	Dr. Smith
3	Michael Lee	CS101	Programming	3	Dr. Smith
3	Michael Lee	PHY301	Physics	4	Dr. Brown

- We'll walk through the process of normalizing a single table with duplicates to 3rd Normal Form (3NF) and explain the different types of relationships that emerge.

Let’s look at an example

StudentID	StudentName	CourseID	CourseName	CourseCredits	InstructorName
1	John Doe	CS101	Programming	3	Dr. Smith
1	John Doe	MAT201	Calculus	4	Prof. Johnson
2	Jane Smith	CS101	Programming	3	Dr. Smith
3	Michael Lee	CS101	Programming	3	Dr. Smith
3	Michael Lee	PHY301	Physics	4	Dr. Brown

- In this table, we can observe data duplication. The student names and course details are repeated for each enrollment. This design violates the principles of normalization and can lead to data anomalies and inconsistencies.

Step 1: First Normal Form (1NF)

- To achieve 1NF, we need to ensure that each column contains atomic values and there are no repeating groups.
- In this case, we can split the table into two separate tables: "Students" and "Enrollments".

Students Table:

StudentID	StudentName
1	John Doe
2	Jane Smith
3	Michael Lee

Enrollments Table:

EnrollmentID	StudentID	CourseID	CourseName	CourseCredits	InstructorName
1	1	CS101	Programming	3	Dr. Smith
2	1	MAT201	Calculus	4	Prof. Johnson
3	2	CS101	Programming	3	Dr. Smith
4	3	CS101	Programming	3	Dr. Smith
5	3	PHY301	Physics	4	Dr. Brown

Step 2: Second Normal Form (2NF)

- To achieve 2NF, we need to ensure that all non-key columns depend on the entire primary key.
- In the "Enrollments" table, the primary key is a combination of "StudentID" and "CourseID".
- However, the course details (CourseName, CourseCredits, InstructorName) depend only on the "CourseID".
- To resolve this, we can split the "Enrollments" table further into "Enrollments" and "Courses" tables.

Enrollments Table:

EnrollmentID	StudentID	CourseID
1	1	CS101
2	1	MAT201
3	2	CS101
4	3	CS101
5	3	PHY301

Courses Table:

CourseID	CourseName	CourseCredits	InstructorName
CS101	Programming	3	Dr. Smith
MAT201	Calculus	4	Prof. Johnson
PHY301	Physics	4	Dr. Brown

Step 3: Third Normal Form (3NF)

- To achieve 3NF, we need to ensure that there are no transitive dependencies. In the "Courses" table, the "InstructorName" depends on the "CourseID" but not directly. It depends on the instructor associated with the course. To resolve this, we can split the "Courses" table into "Courses" and "Instructors" tables.

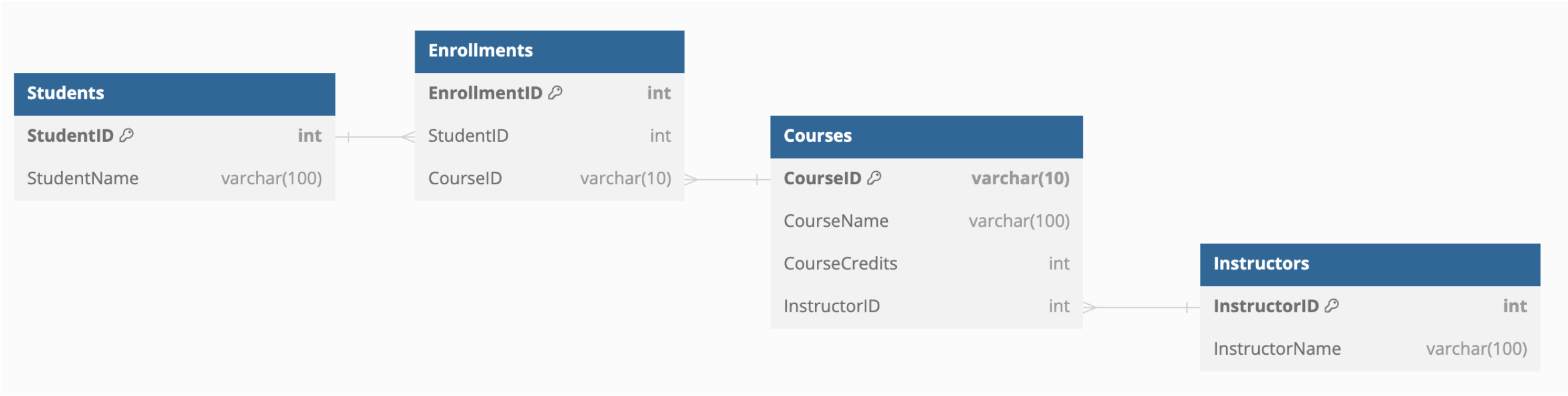
Courses Table:

CourseID	CourseName	CourseCredits	InstructorID
CS101	Programming	3	1
MAT201	Calculus	4	2
PHY301	Physics	4	3

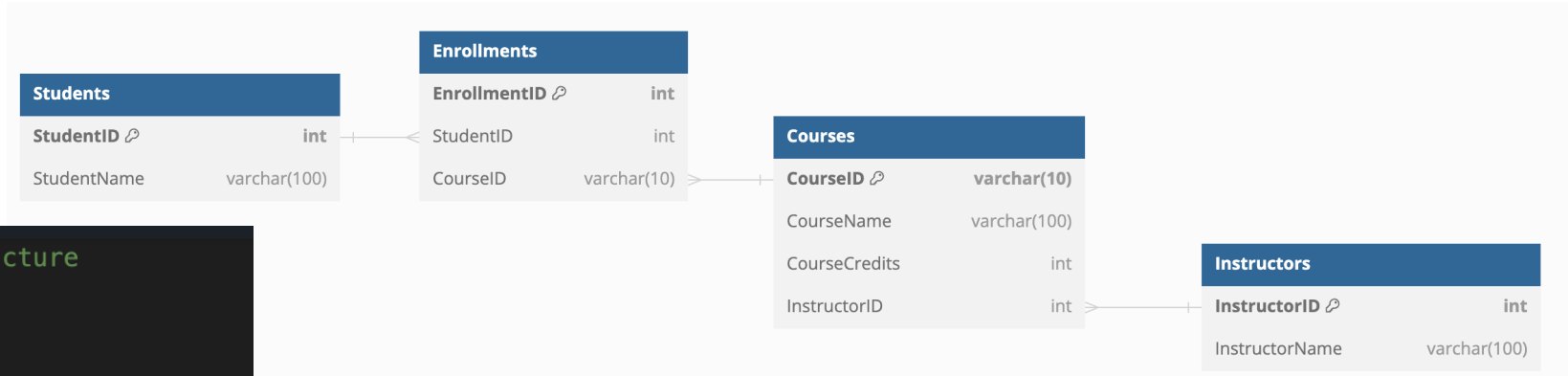
Instructors Table:

InstructorID	InstructorName
1	Dr. Smith
2	Prof. Johnson
3	Dr. Brown

The 3NF Schema



The 3NF Schema – Created with DBML




```
// Use DBML to define your database structure
// Docs: https://dbml.dbdiagram.io/docs

Table Students {
  StudentID int [pk]
  StudentName varchar(100)
}

Table Courses {
  CourseID varchar(10) [pk]
  CourseName varchar(100)
  CourseCredits int
  InstructorID int [ref: > Instructors.InstructorID]
}

Table Instructors {
  InstructorID int [pk]
  InstructorName varchar(100)
}

Table Enrollments {
  EnrollmentID int [pk]
  StudentID int [ref: > Students.StudentID]
  CourseID varchar(10) [ref: > Courses.CourseID]
}
```

- Represent the table in Database Markup Language (DBML)
- Rendered visually in https://www.dbdiagram.io
- You could also use Kroki.io or PHPMyAdmin or any other diagramming tool.

Week 5: Todo

- Weekly RiPPLE task is due in Week 5 on Friday at 3pm
- Complete Week 5 Lab 5
  - You'll learn how to use Bootstrap CSS to make a better UI for the Resume Builder app
- Continue with your Design Document Assessment Item
  - You should now be finishing the database design
  - You should be starting the HTML Mockups

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA
CREATE CHANGE

Design Document Assessment Q&A

Thank you