INFS3202/7202 – Web Information Systems

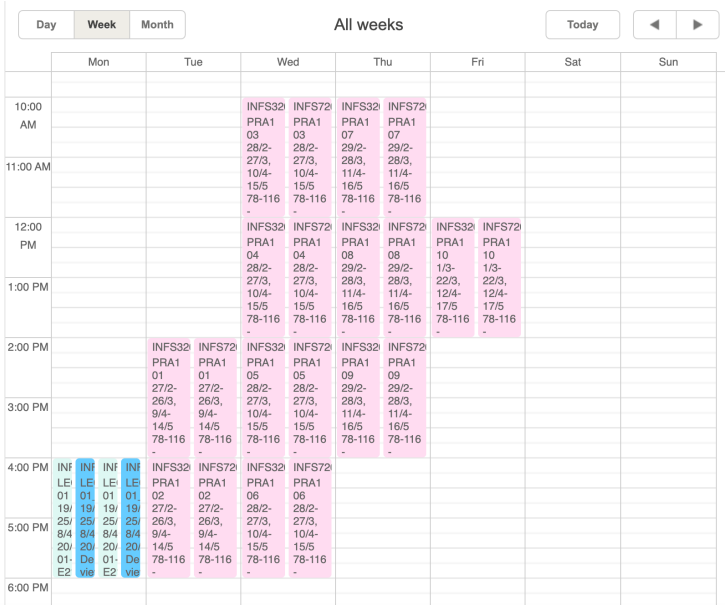
Lecture Week 6: MVC – Advanced CodeIgniter

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

Contents

Course Updates

Issue/Feedback	Change
Friday 29 March is a Public Holiday	• Students in Fridays Lab can attend any other Lab Tues - Thursday
RiPPLE Weekly Task	• Due on Thursday 28 March at 3pm
Mid-Semester Break	• Next week is break week
Design Document Assessment Item	• Due Monday of Week 7



Tips for Design Document Assessment Item

A1 Design Document Assessment Item

- Worth 20% of overall grade

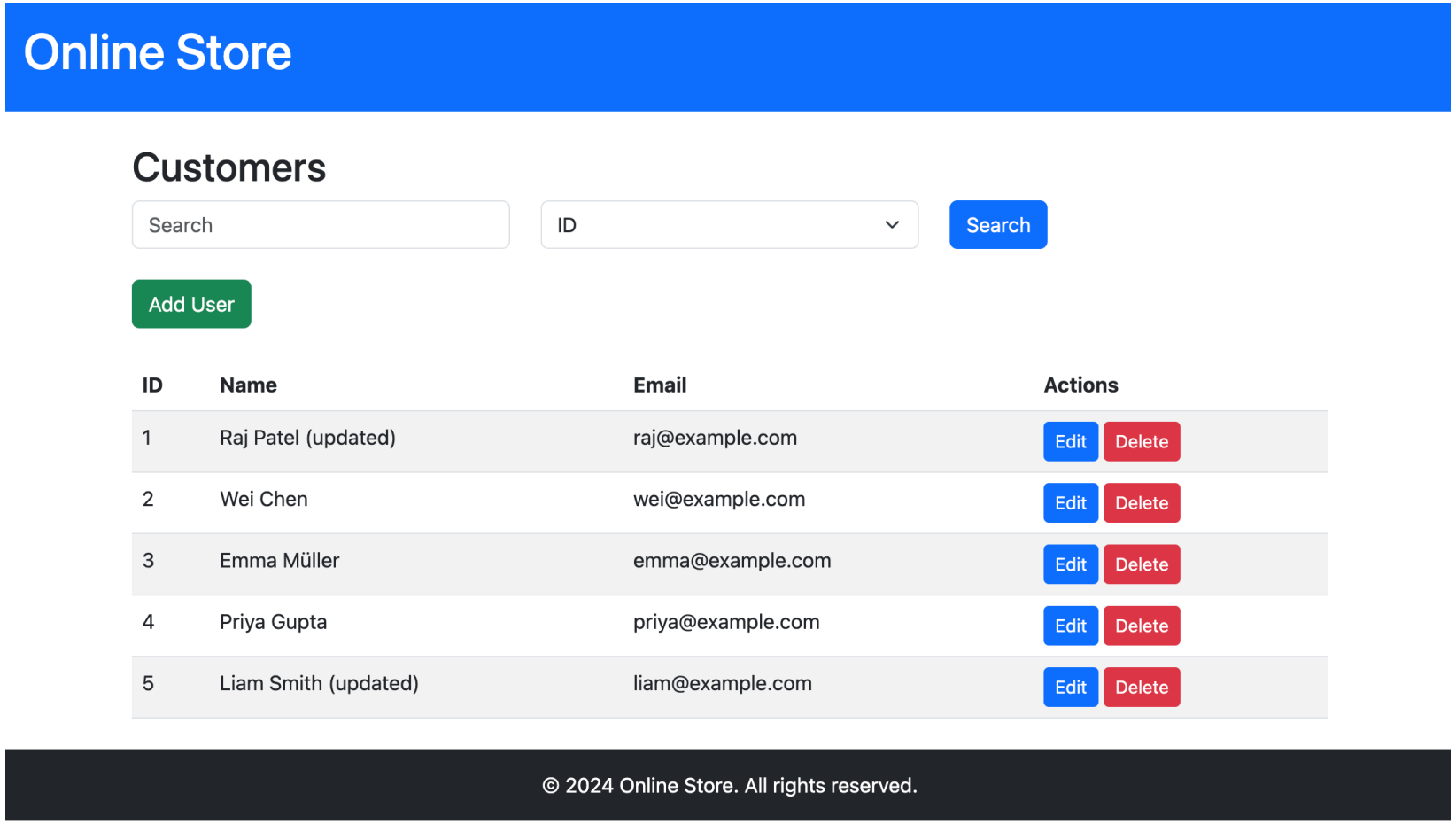
- **Database Design Tips:**
  - Remember you are building a Software as a Service (SaaS)
  - This means that the database design needs to support multiple clients using your functionality
  - Review the Database Design Walkthrough from the Week 5 Lecture if you are not sure how to normalise the database to 3rd Normal Form.

- **HTML Mockups Tips:**
  - Lab 5 is very useful if you don't know how to make a HTML Mockup
  - You only need 4-5 of the main screen of your application

- If you still have question or need help please email me (a.bakharia1@uq.edu.au) for an appointment to meet
- After submission in Week 7 your next steps are to create the database in MySQL and make the HTML into a base template and views.

Optimised CRUD (Customers App)

code in week6_code/customers_optimised



- Let’s reduce code redundancy
  - Single Controller Method to add and edit the form
- Add server side validation

Customers App - Routes

cidemo > app > Config > Routes.php

```php
<?php

use CodeIgniter\Router\RouteCollection;

/**
 * @var RouteCollection $routes
 */

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

// Display and process add
$routes->match(['get', 'post'], '/addeditform', 'CustomerController::addeditform');
// Display and process edit
$routes->match(['get', 'post'], '/addeditform/(:num)', 'CustomerController::addeditform/$1');

$routes->get('/delete/(:num)', 'CustomerController::delete/$1');

```

Customers App - Controller

cidemo > app > Controllers > CustomerController.php

```php
    public function addeditform($id = null)
    {
        $model = new \App\Models\Customers();
        $customer = $id ? $model->find($id) : [];

        if ($id && !$customer) {
            throw new PageNotFoundException('Customer not found');
        }

        $data = [
            'title' => $id ? 'Edit Customer' : 'Add Customer',
            'customer' => $customer,
        ];

        if ($this->request->getMethod() === 'post') {
            $rules = [
                'name' => 'required|min_length[3]',
                'email' => 'required|valid_email',
            ];

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

            if ($this->validate($rules)) {
                if ($id) {
                    $model->update($id, $customerData);
                    $this->session->setFlashdata('success', 'Customer updated successfully');
                } else {
                    $model->save($customerData);
                    $this->session->setFlashdata('success', 'Customer added successfully');
                }

                return redirect()->to('customers');
            } else {
                $data['validation'] = $this->validator;
                $data['customer'] = $customerData;
            }
        }

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

https://codeigniter.com/user_guide/libraries/validation.html

Customers App – AddEditForm View

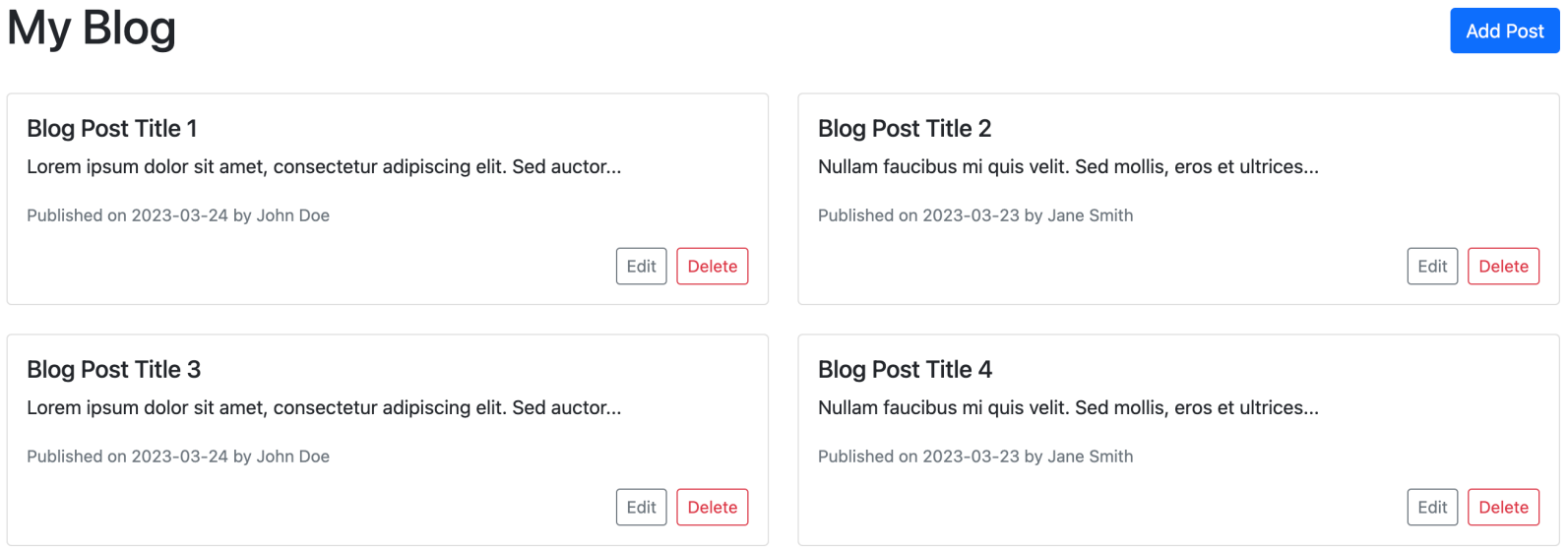
cidemo > app > Views > customer_form.php

```php
<?= $this->extend('base_template') ?>
<?= $this->section('content') ?>
<h2><?= $title ?></h2>

<?php if (isset($validation)): ?>
    <div class="alert alert-danger">
        <?= $validation->listErrors() ?>
    </div>
<?php endif; ?>

<form method="post" action="<?= base_url('addeditform/' . (isset($customer['id']) ? $customer['id'] : '')) ?>">
    <div class="mb-3">
        <label for="name" class="form-label">Name</label>
        <input type="text" name="name" id="name" class="form-control" value="<?= isset($customer['name']) ? $customer['name'] : '' ?>" required>
    </div>
    <div class="mb-3">
        <label for="email" class="form-label">Email</label>
        <input type="email" name="email" id="email" class="form-control" value="<?= isset($customer['email']) ? $customer['email'] : '' ?>" required>
    </div>
    <button type="submit" class="btn btn-primary"><?= isset($customer['id']) ? 'Update' : 'Save' ?></button>
</form>
<?= $this->endSection() ?>
```

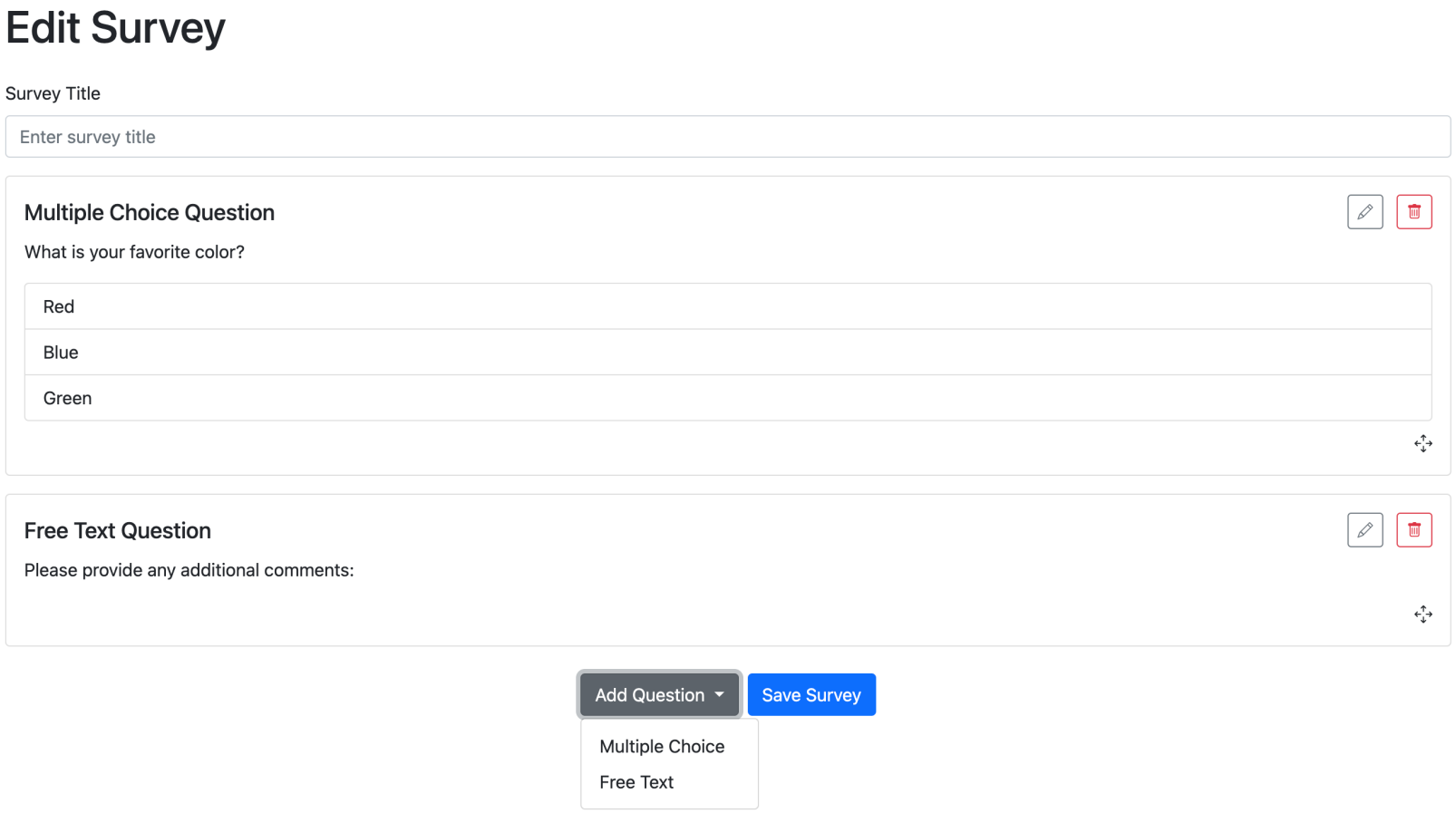
Example CRUD UI’s

(All created with Bootstrap 5)

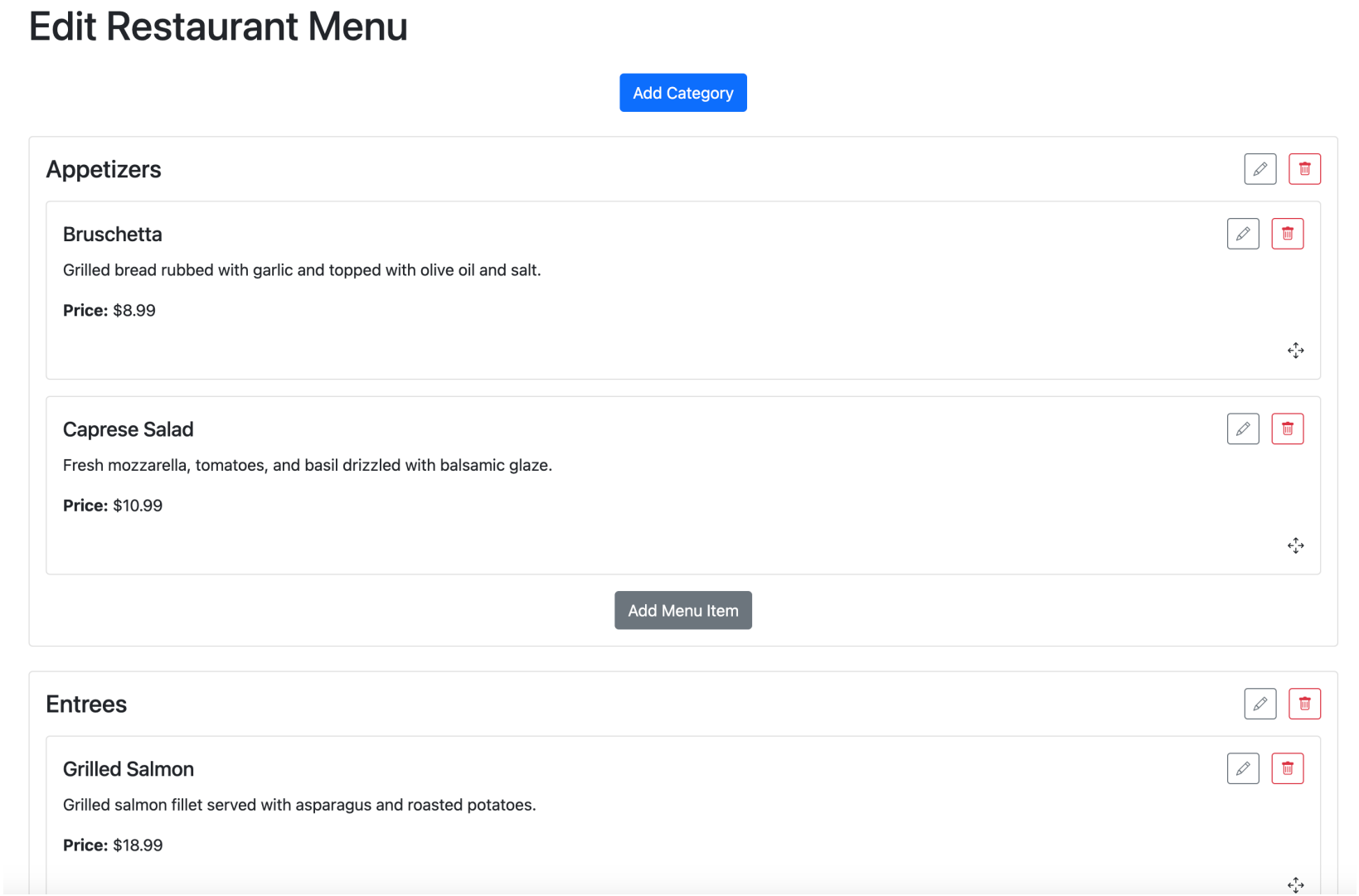
Example CRUD UI's – Blog Creation



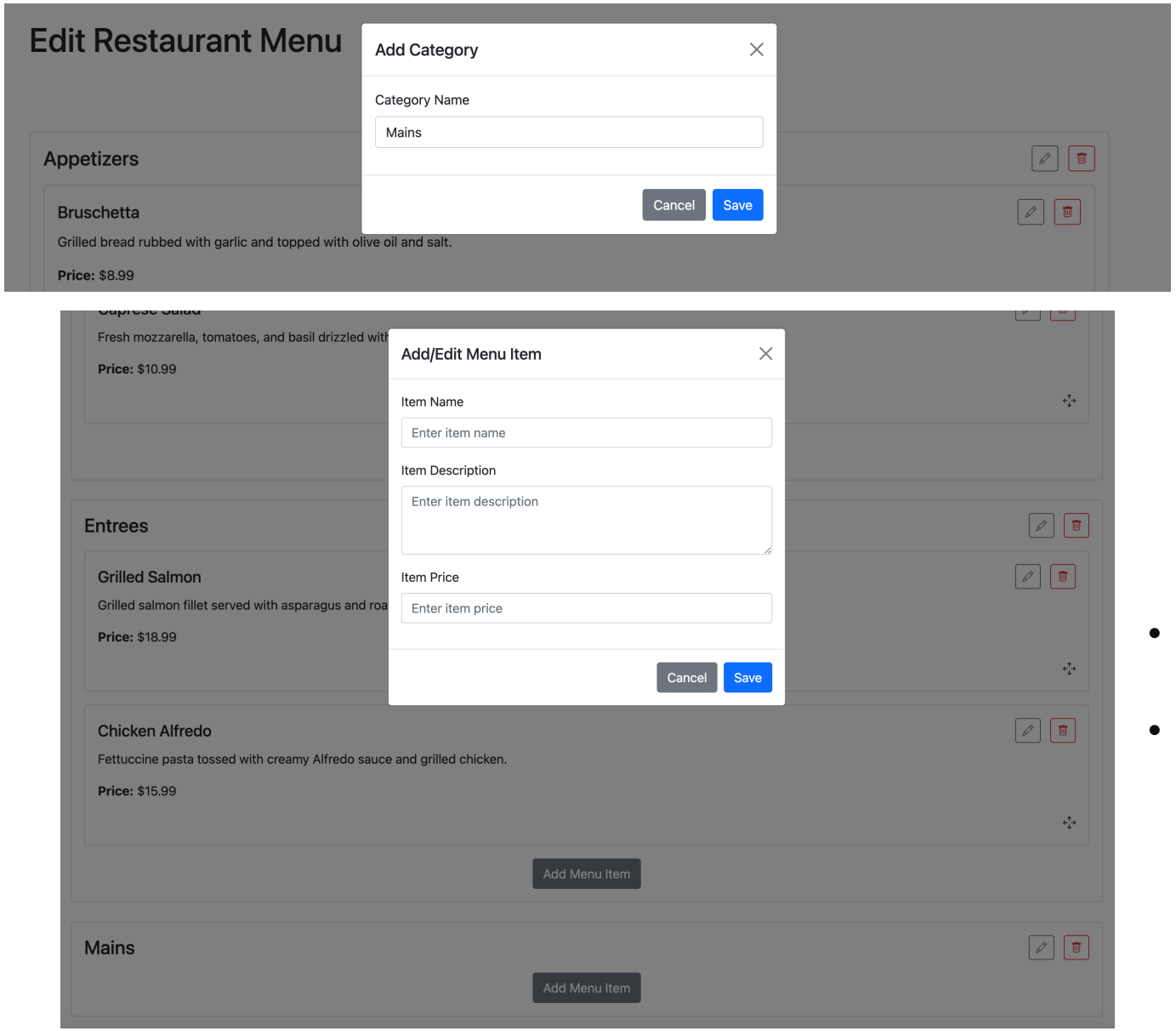
Example CRUD UI's – Survey Builder



Example CRUD UI's – Menu Builder



Example CRUD UI's – Menu Builder



- Uses Javascript to avoid reloading the page.
- Will Learn how to do this in Week 8

Caching with CodeIgniter

code in week6_code/advanced_ci

What is caching?

- Caching is a technique used to store copies of files or data results in a temporary storage location (cache), so future requests for those files or data can be served faster.
- In web development, caching can refer to storing parts of web pages or views, database query results, or even computational results to improve performance and reduce the load on the system.

- **Why Do We Need It?**

  - **Performance Enhancement:**
    By reducing the need to perform time-consuming operations (like database queries or complex calculations) for every request, caching significantly speeds up the response time of applications.
  - **Reduced Load:**
    It decreases the load on the server, as cached content is served directly without going through the entire application logic or database querying process again.
  - **Improved User Experience:**
    Faster response times contribute to a smoother and more responsive user experience.
  - **Scalability:**
    Caching helps in scaling applications by managing resource utilization more efficiently, allowing the system to handle more users without requiring proportional increases in hardware or processing power.

What type of Caching does CodeIgniter Support?

CodeIgniter provides a simple yet flexible caching mechanism that supports different types of caching:

- **Page Caching:**
  This method caches the entire output of a page. When a request is made, CodeIgniter checks if a cached version of the page exists and serves it directly, bypassing the normal system execution.

- **Querystring Caching:**
  Uses the combination of variables in the URL querystring and saves versions to the cache.

Cache Config

- **Location of Cache Settings:**
  The primary configuration file for cache settings in CodeIgniter is located at app/Config/Cache.php. Specify the default cache handler eg file or redis etc.

app > Config > Cache.php

```php
<?php

namespace Config;

use CodeIgniter\Cache\CacheInterface;
use CodeIgniter\Cache\Handlers\DummyHandler;
use CodeIgniter\Cache\Handlers\FileHandler;
use CodeIgniter\Cache\Handlers\MemcachedHandler;
use CodeIgniter\Cache\Handlers\PredisHandler;
use CodeIgniter\Cache\Handlers\RedisHandler;
use CodeIgniter\Cache\Handlers\WincacheHandler;
use CodeIgniter\Config\BaseConfig;

class Cache extends BaseConfig
{
    /**
     * ---------------------------------------------------------------------
     * Primary Handler
     * ---------------------------------------------------------------------
     *
     * The name of the preferred handler that should be used. If for some reason
     * it is not available, the $backupHandler will be used in its place.
     */
    public string $handler = 'file';

    /**
     * ---------------------------------------------------------------------
     * Backup Handler
     * ---------------------------------------------------------------------
     *
     * The name of the handler that will be used in case the first one is
     * unreachable. Often, 'file' is used here since the filesystem is
     * always available, though that's not always practical for the app.
     */
    public string $backupHandler = 'dummy';

    /**
     * ---------------------------------------------------------------------
     * Cache Directory Path
     * ---------------------------------------------------------------------
     *
     * The path to where cache files should be stored, if using a file-based
```

Caching a View

app > Controllers > Home.php

```php
<?php

namespace App\Controllers;

class Home extends BaseController
{
    public function index(): string
    {
        return view('welcome_message');
    }

    public function cached(): string
    {

        $this->cachePage(60); // Cache the output for 60 seconds

        return view('cacheexampleview');
    }
}

```

app > Views > cacheexampleview.php

```php
<!DOCTYPE html>
<html>
<head>
    <title>Current Time</title>
</head>
<body>
    <h1>Current Time Display</h1>
    <p>The current time is: <?php echo date('H:i:s'); ?></p>
</body>
</html>

```

https://www.codeigniter.com/user_guide/general/caching.html

Where is the cached file stored?

writable > cache > 3a7563881fdc9a84ebf42b3f76685e4e > ...

```
a:3:{s:4:"time";i:1711187321;s:3:"ttl";i:60;s:4:"data";s:22568:"a:2:{s:7:"headers"
<!DOCTYPE html>
<html>
<head>
<script  id="debugbar_loader" data-time="1711187321.032992" src="http://localhost:
void 0===window.kintRich&&(window.kintRich=function(){"use strict";var l={selectTe
void 0===window.kintMicrotimeInitialized&&(window.kintMicrotimeInitialized=1,windo
</script><style class="kint-rich-style">.kint-rich{font-size:13px;overflow-x:auto;
</style>

    <title>Current Time</title>
</head>
<body>
    <h1>Current Time Display</h1>
    <p>The current time is: 09:48:41</p>
</body>
</html>

<!-- DEBUG-VIEW ENDED 1 APPPATH/Views/cacheexampleview.php -->
";}";}
```

Cached files are saved in the writable/cache folder

CodeIgniter Spark Cache Commands

- **Get cache information**
  - `php spark cache:info`
- **Clear cache**
  - `php spark cache:clear`

```
uqabakh1@uqabakh1-9893 advanced_ci % php spark cache:info

CodeIgniter v4.4.6 Command Line Tool - Server Time: 2024-03-23 09:53:25 UTC+00:00

+----------------------------------+------------------------------------------------------------+-----------+---------------------+
| Name                             | Server Path                                                | Size      | Date                |
+----------------------------------+------------------------------------------------------------+-----------+---------------------+
| index.html                       | ROOTPATH/writable/cache/index.html                         | 131 Bytes | 2024-02-24 02:42:57 |
| 3a7563881fdc9a84ebf42b3f76685e4e | ROOTPATH/writable/cache/3a7563881fdc9a84ebf42b3f76685e4e | 22.1 KB   | 2024-03-23 09:53:11 |
+----------------------------------+------------------------------------------------------------+-----------+---------------------+

uqabakh1@uqabakh1-9893 advanced_ci % php spark cache:clear

CodeIgniter v4.4.6 Command Line Tool - Server Time: 2024-03-23 09:53:55 UTC+00:00

Cache cleared.
```

File Uploads

The importance of supporting file uploads

- File uploads are a critical component of many web applications, from social media platforms where users share images and videos, to business applications that handle documents, spreadsheets, and other business-critical files.

  - **Personalization:** Allows users to upload avatars, backgrounds, or other personal images, enhancing their engagement and personal connection to the application.
  - **Content Sharing:** Enables users to share content, such as photos, videos, and documents, facilitating collaboration and interaction within the application.
  - **Dynamic Content:** File uploads are essential for content management systems (CMS), enabling administrators and users to update website content, including text, images, and videos.
  - **Data Import:** Enables efficient data import processes, allowing users to upload files containing bulk data, which can be processed and integrated into the application's database.

Processing a File Upload (part 1)

- Create Routes
  - Same /uploads route will handle get and post
  - Get will display the upload form
  - Post will accept the posted file

app > Config > Routes.php

```php
<?php

use CodeIgniter\Router\RouteCollection;

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

$routes->get('/cached', 'Home::cached');

$routes->get('/testviewcell', 'Home::testviewcell');

$routes->get('/upload', 'FileUploadController::index');
$routes->post('/upload', 'FileUploadController::upload');

```

Processing a File Upload (part 2)

- Create the View
  - Use the DropZone Javascript Library
  - A CSRF must be included in the form to make sure the form post is from the same server.

Dropzone File Upload

Drop files here to upload

app > Views > uploadform.php

```
<!DOCTYPE html>
<html>
<head>
    <title>Dropzone File Upload</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.9.2/min/dropzone.min.css">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/dropzone/5.9.2/min/dropzone.min.js"></script>
    <style>…
    </style>
</head>
<body>
    <h2>Dropzone File Upload</h2>

    <form action="<?= base_url('upload'); ?>" class="dropzone" id="myDropzone">
        <?= csrf_field() ?>
    </form>

    <div id="message" class="message"></div>

    <script>
        Dropzone.options.myDropzone = {
            paramName: "file",
            maxFilesize: 2, // MB
            acceptedFiles: ".jpg,.jpeg,.png,.gif",
            init: function() {
                this.on("success", function(file, response) {
                    if (response.success) {
                        showMessage("File uploaded successfully!", "success");
                    } else {
                        showMessage("File upload failed!", "error");
                    }
                });
                this.on("error", function(file, errorMessage) {
                    showMessage("File upload error: " + errorMessage, "error");
                });
            }
        };

        function showMessage(message, type) {
            var messageElement = document.getElementById("message");
            messageElement.textContent = message;
            messageElement.className = "message " + type;
        }
    </script>
</body>
</html>
```

Processing a File Upload (part 3)

- Create the Controller
  - The file is saved to /writable/uploads with a random filename
  - A JSON response needs to be returned
  - You will need to manage where uploads are stored for your users and how to secure access

app > Controllers > FileUploadController.php

```php
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

class FileUploadController extends Controller
{
    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
        helper('url');

    }

    public function index()
    {
        return view('uploadform');
    }

    public function upload()
    {
        $file = $this->request->getFile('file');

        if ($file->isValid() && !$file->hasMoved()) {
            $newName = $file->getRandomName();
            $file->move(WRITEPATH . 'uploads', $newName);

            // Store the file information in the database or perform other operations

            return $this->response->setJSON(['success' => true]);
        } else {
            return $this->response->setJSON(['success' => false]);
        }
    }
}
```

View Cells

Creating a View Cell

- A view cell is a re-usable code fragment/component that can take parameters and render HTML

- Very useful for re-usable elements eg Latest News, Ad's or Alerts

- **Create a View cell**
  - `php spark make:cell AlertMessageCell`

```
uqabakh1@uqabakh1-9893 advanced_ci % php spark make:cell AlertMessageCell

CodeIgniter v4.4.6 Command Line Tool - Server Time: 2024-03-23 10:15:20 UTC+00:00

File created: APPPATH/Cells/AlertMessageCell.php

File created: APPPATH/Cells/alert_message.php
```

Creating a View Cell

app > Cells > AlertMessageCell.php

```php
<?php

// app/Cells/AlertMessageCell.php

namespace App\Cells;

use CodeIgniter\View\Cells\Cell;

class AlertMessageCell extends Cell
{
    public $type;
    public $message;
}

```

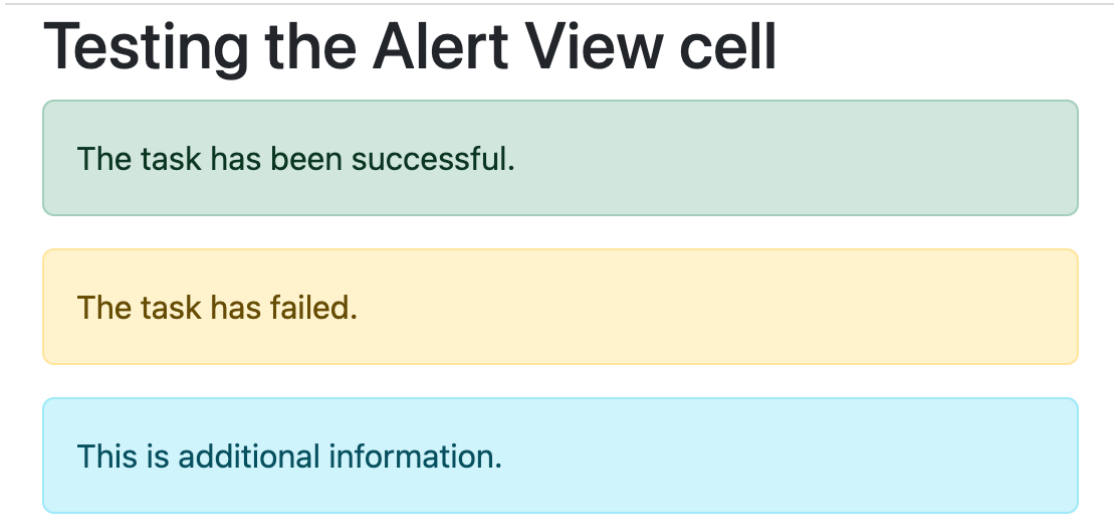
app > Cells > alert_message.php

```php
<div class="alert alert-<?= esc($type, 'attr') ?>" role="alert">
    <?= esc($message) ?>
</div>

```

Using a View Cell

app > Views > testviewcell.php

```
1   <?= $this->extend('template') ?>
2   <?= $this->section('content') ?>
3
4   <h1>Testing the Alert View cell</h1>
5
6   <?= view_cell('AlertMessageCell', 'type=success, message=The task has been successful.') ?>
7
8   <?= view_cell('AlertMessageCell', 'type=warning, message=The task has failed.') ?>
9
10  <?= view_cell('AlertMessageCell', 'type=info, message=This is additional information.') ?>
11
12
13  <?= $this->endSection() ?>
14
```



Package Management with Composer (Installing a QR Code Library)

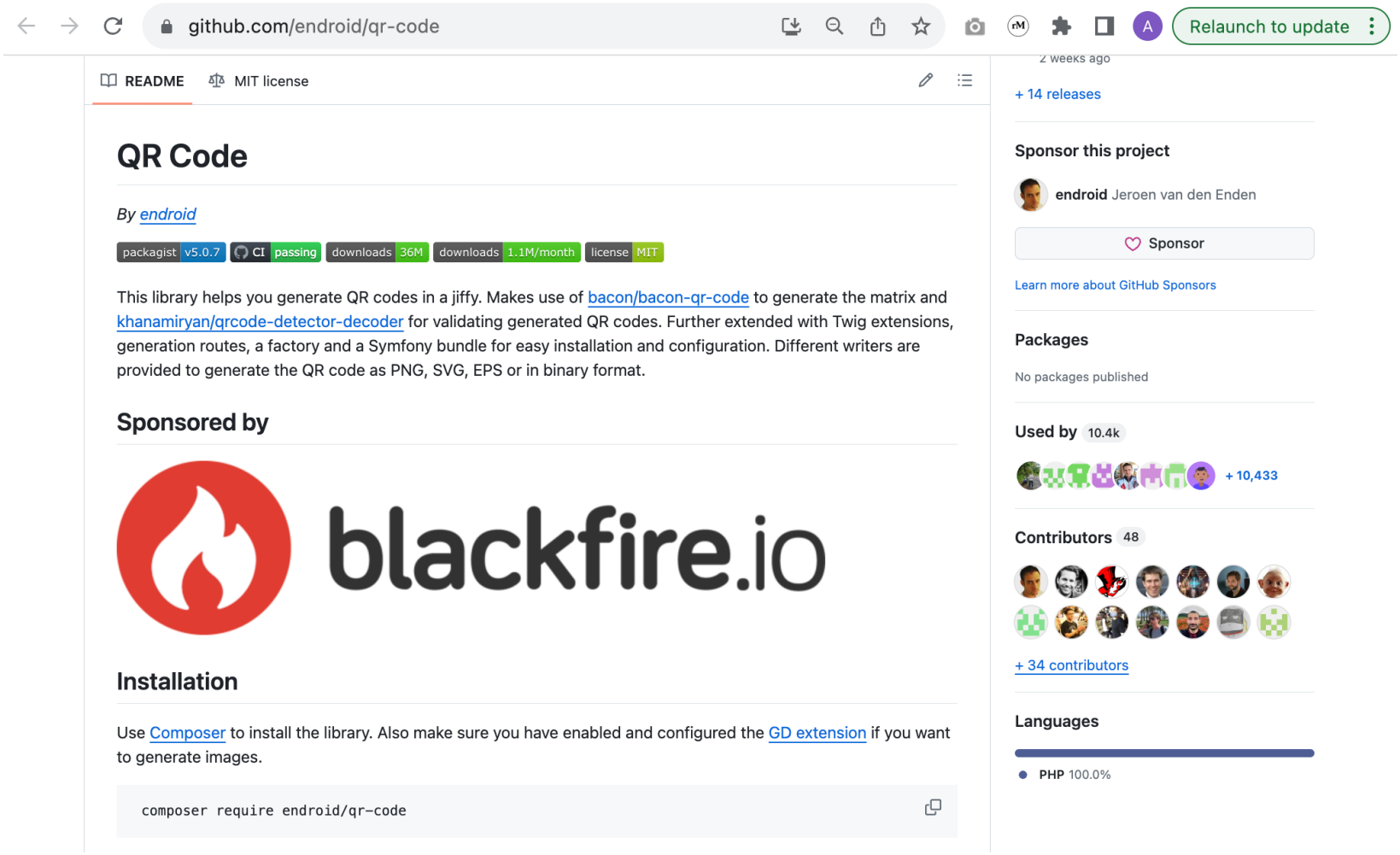
What is Composer?

- **You currently use composer to create a new CodeIgniter Project.**

Composer is also a Package Manager

- **Manage Packages**: Composer allows you to declare the libraries your project depends on and manages (installs/updates) them for you.
- **CLI Tool**: Composer operates through a command-line interface, making it accessible for automation and integration with development, testing, and deployment workflows.
- **composer.lock**: When you install dependencies, Composer creates a composer.lock file, locking your project to specific versions of packages. This ensures that all team members and production environments use the same versions, leading to consistent environments and avoiding "works on my machine" issues.
- **Update Control**: You can update dependencies individually or collectively to newer versions as needed, according to the constraints defined in composer.json, while composer.lock ensures that these updates are tracked and consistent across all environments.

Using Composer to Install a QR Code Library



https://github.com/endroid/qr-code

Using Composer to Install a QR Code Library

```
uqabakh1@uqabakh1-9893 advanced_ci % composer require endroid/qr-code
./composer.json has been updated
Running composer update endroid/qr-code
Loading composer repositories with package information
Updating dependencies
Lock file operations: 3 installs, 0 updates, 0 removals
  - Locking bacon/bacon-qr-code (2.0.8)
  - Locking dasprid/enum (1.0.5)
  - Locking endroid/qr-code (5.0.7)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 3 installs, 0 updates, 0 removals
  - Downloading dasprid/enum (1.0.5)
  - Downloading bacon/bacon-qr-code (2.0.8)
  - Downloading endroid/qr-code (5.0.7)
  - Installing dasprid/enum (1.0.5): Extracting archive
  - Installing bacon/bacon-qr-code (2.0.8): Extracting archive
  - Installing endroid/qr-code (5.0.7): Extracting archive
4 package suggestions were added by new dependencies, use `composer suggest` to see details.
Generating optimized autoload files
29 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found.
Using version ^5.0 for endroid/qr-code
```

https://github.com/endroid/qr-code

Using Composer to Install a QR Code Library

```
uqabakh1@uqabakh1-9893 advanced_ci % composer require endroid/qr-code
./composer.json has been updated
Running composer update endroid/qr-code
Loading composer repositories with package information
Updating dependencies
Lock file operations: 3 installs, 0 updates, 0 removals
  - Locking bacon/bacon-qr-code (2.0.8)
  - Locking dasprid/enum (1.0.5)
  - Locking endroid/qr-code (5.0.7)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 3 installs, 0 updates, 0 removals
  - Downloading dasprid/enum (1.0.5)
  - Downloading bacon/bacon-qr-code (2.0.8)
  - Downloading endroid/qr-code (5.0.7)
  - Installing dasprid/enum (1.0.5): Extracting archive
  - Installing bacon/bacon-qr-code (2.0.8): Extracting archive
  - Installing endroid/qr-code (5.0.7): Extracting archive
4 package suggestions were added by new dependencies, use `composer suggest` to see details.
Generating optimized autoload files
29 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found.
Using version ^5.0 for endroid/qr-code
```

https://github.com/endroid/qr-code

Using Composer to Install a QR Code Library

app > Controllers > Home.php

```php
<?php

namespace App\Controllers;
use Endroid\QrCode\QrCode;
use Endroid\QrCode\Writer\PngWriter;

class Home extends BaseController
{
    public function index(): string
    {…
    }

    public function cached(): string
    {…
    }

    public function testviewcell(): string
    {…
    }

    public function generateQRCode()
    {
        $qr_code = QrCode::create('https://www.uq.edu.au');
        $writer = new PngWriter;
        $result = $writer->write($qr_code);

        // Set the response headers
        $this->response->setHeader('Content-Type', $result->getMimeType());

        // Output the QR code image directly
        $this->response->setBody($result->getString());

        // Return the response
        return $this->response->send();
    }
}
```

- We return an image with a png image mimetype.
- You can use the url inside an image tag and set the src to the url to get the image displayed on a webpage
- There are also js QR code generators https://davidshimjs.github.io/qrcodejs/

Sessions

Sessions

- Sessions are a fundamental aspect of web development, enabling web applications to store and access user data across multiple requests.

- **User Data Storage:**
  Sessions provide a way to persist user data across multiple web pages or requests. Unlike cookies, session data is stored on the server.
- **Unique Identifiers:**
  Each user is identified through a unique session ID, typically sent to the user's browser via a cookie. This ID is then used to retrieve session data on subsequent requests.

Why Use Sessions?

- **State Management:**
  HTTP is a stateless protocol, meaning it doesn't retain information between requests. Sessions enable a stateful experience, allowing applications to "remember" user actions and preferences.

Sessions in CodeIgniter

- **Framework Integration**:
  - CodeIgniter's Session Library provides a more robust and flexible way to handle session data compared to native PHP sessions.
  - It supports various storage drivers (files, database, Redis, Memcached) for session data.

- **Key Features**
  - **Flashdata**: Special session data that is only available for the next server request, and is automatically deleted afterwards. Ideal for one-time messages, like form submission success or error messages.
  - **Tempdata**: Similar to flashdata, but allows for specifying a time limit (in seconds) for how long the data should be available in the session.
  - **Regeneration of Session ID**: CodeIgniter automatically regenerates the session ID periodically to prevent session fixation attacks.

  - Session data is simply an array associated with a particular session ID (cookie).

https://www.codeigniter.com/user_guide/libraries/sessions.html

Login Example using Sessions - Routes

app > Config > Routes.php

```php
<?php

use CodeIgniter\Router\RouteCollection;

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

$routes->get('/cached', 'Home::cached');

$routes->get('/testviewcell', 'Home::testviewcell');

$routes->get('/upload', 'FileUploadController::index');
$routes->post('/upload', 'FileUploadController::upload');

$routes->get('/qrcode', 'Home::generateQRCode');

$routes->match(['get', 'post'], '/login', 'AuthController::login');
$routes->get('/logout', 'AuthController::logout');

$routes->get('/dashboard', 'DashboardController::index');
```

Login Example using Sessions - AuthController

app > Controllers > AuthController.php

```php
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

class AuthController extends Controller
{
    public function login()
    {
        if ($this->request->getMethod() === 'post') {
            $email = $this->request->getPost('email');
            $password = $this->request->getPost('password');

            // Perform authentication logic here
            // Mostly likely check against a database
            // For simplicity we have hardcoded values
            if ($email === 'user@example.com' && $password === 'password') {
                // Authentication successful, set session data
                $sessionData = [
                    'user_id' => 1,
                    'user_email' => $email,
                    'logged_in' => true
                ];
                session()->set($sessionData);

                return redirect()->to('dashboard');
            } else {
                // Authentication failed, display error message
                session()->setFlashdata('error', 'Invalid email or password');
                return redirect()->back();
            }
        }

        return view('login');
    }

    public function logout()
    {
        session()->destroy();
        return redirect()->to('/login');
    }
}
```

Login Example using Sessions – Dashboard Controller

app > Controllers > DashboardController.php

```php
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

class DashboardController extends Controller
{
    public function index()
    {
        // Check if the user is logged in
        if (!session()->get('logged_in')) {
            return view('not_authorized');
        }

        // User is logged in, display the dashboard
        $data = [
            'user_email' => session()->get('user_email')
        ];

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

Login Example using Sessions – Login view

app > Views > login.php

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

<h2>Login</h2>

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

<form action="<?= base_url('/login'); ?>" method="post">
    <?= csrf_field() ?>
    <div class="mb-3">
        <label for="email" class="form-label">Email:</label>
        <input type="email" name="email" id="email" class="form-control" value="user@example.com" required>
    </div>
    <div class="mb-3">
        <label for="password" class="form-label">Password:</label>
        <input type="password" name="password" id="password" class="form-control" value="password" required>
    </div>
    <button type="submit" class="btn btn-primary">Login</button>
</form>

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

```

Login Example using Sessions – Dashboard view

app > Views > dashboard.php

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

<h2>Welcome to the Dashboard</h2>

<p>You are logged in as: <?= $user_email ?></p>

<p><a href="<?= base_url('logout'); ?>">Logout</a></p>

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

```

Login Example using Sessions – Dashboard view

app > Views > not_authorized.php

```
1   <?= $this->extend('template') ?>
2   <?= $this->section('content') ?>
3
4   <h2>Not Authorized</h2>
5
6   <p>You do not have permission to access this page.</p>
7
8   <p><a href="<?= base_url('login') ?>">Login</a></p>
9
10  <?= $this->endSection() ?>
```

Build your own Auth using Sessions for learning…
But I recommend using an Auth library
such as CodeIgniter Shield …
unless you want to sign in using Github or Google!

Authentication vs Authorization

Authentication vs Authorisation

- Authentication and authorization are fundamental security processes in the context of web and application development.
- While they are closely related and often used together, they serve distinct purposes.
- Authentication is about verifying identity (who you are), while authorization is about granting access to resources (what you are allowed to do).
- Authentication always precedes authorization. A system must first recognize who you are before it can determine what you are allowed to access.
- Authorization is managed through settings, configurations, or roles defining access levels and permissions.

Authentication

- **Identity Verification**:
  Authentication is the process of verifying who a user is. It involves confirming the identity of a user attempting to access a system or application.

- **Credential Check**:
  Typically involves validating user credentials against a database or through other means. Common methods include usernames and passwords, biometric verification, OTPs (One-Time Passwords), or security tokens.

- **First Step in Security Process**:
  Authentication is the initial step in the security process, determining whether a user is who they claim to be before granting access to a system.

- **Examples**:
  Logging into a web application, entering a PIN at an ATM, or using a fingerprint to unlock a smartphone.

Authorisation

- **Access Control**:
  Authorization occurs after authentication and determines what resources a user is allowed to access and what operations they are permitted to perform.

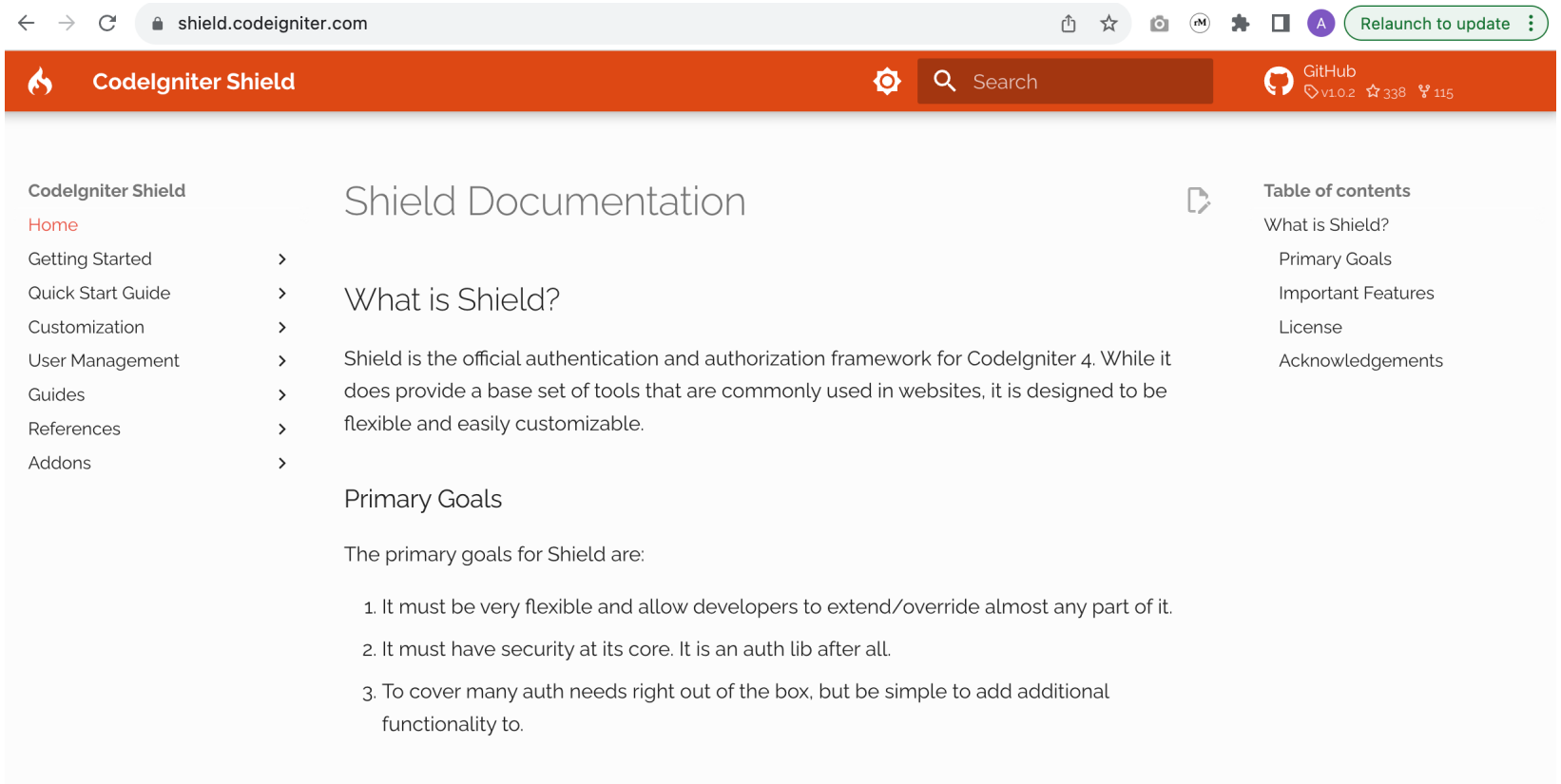
- **Role-Based Access**:
  Often implemented using roles assigned to authenticated users, which dictate access levels or permissions within a system or application.

- **Policy Enforcement**:
  Involves enforcing policies that control access to resources based on roles, clearance levels, or other attributes associated with the authenticated user.

- **Examples**:
  A user (authenticated) is allowed to view a document but not edit it (authorization), or an employee (authenticated) can access employee-only areas within a company's website (authorization).

Using CodeIgniter Shield

CodeIgniter Shield



https://shield.codeigniter.com/

Why use CodeIgniter Shield?

- Built-in features: Shield provides a comprehensive set of authentication and authorization features out of the box, saving development time and effort.
- Security: Shield follows security best practices and provides built-in protection against common vulnerabilities like CSRF, XSS, and SQL injection.
- Customization: Shield offers a flexible and customizable architecture, allowing you to easily extend and adapt it to your specific application requirements.
- User management: Shield includes user management functionalities such as user registration, login, logout, password reset, and email verification.
- Role-based access control: Shield supports role-based access control (RBAC) to restrict access to certain parts of your application based on user roles and permissions.

How to setup CodeIgniter Shield?

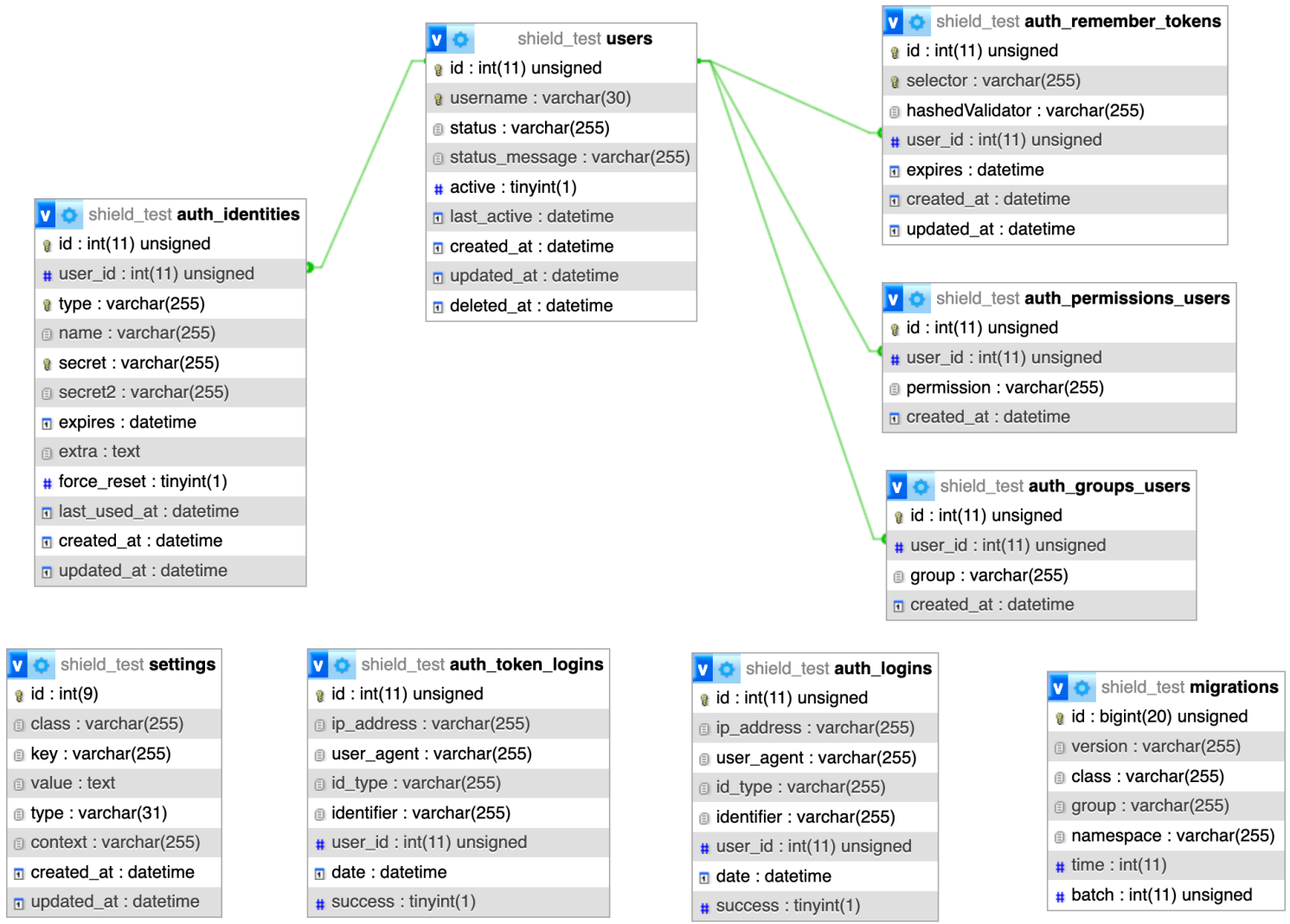
- Use composer to create a CI project
  ```
  composer create-project codeigniter4/appstarter ci_shield
  ```
- Create a new .env file
  - cp env .env
  - Setup base url and database settings
- Use composer to install Shield
  ```
  composer require codeigniter4/shield
  ```
- Run the shield spark command
  ```
  php spark shield:setup
  ```

How to setup CodeIgniter Shield?

```
uqabakh1@uqabakh1-9893 ci_shield % php spark shield:setup

CodeIgniter v4.4.6 Command Line Tool - Server Time: 2024-03-24 00:26:48 UTC+00:00

  Created: APPPATH/Config/Auth.php
  Created: APPPATH/Config/AuthGroups.php
  Created: APPPATH/Config/AuthToken.php
  Updated: APPPATH/Config/Autoload.php
  Updated: APPPATH/Config/Routes.php
  Updated: We have updated file 'APPPATH/Config/Security.php' for security reasons.
  The required Config\Email::$fromEmail is not set. Do you set now? [y, n]: y
  What is your email? : a.bakharia1@uq.edu.au
  The required Config\Email::$fromName is not set. Do you set now? [y, n]: y
  What is your name? : Aneesha Bakharia
  Updated: APPPATH/Config/Email.php
  Run `spark migrate --all` now? [y, n]: y
Running all new migrations...
        Running: (CodeIgniter\Shield) 2020-12-28-223112_CodeIgniter\Shield\Database\Migrations\CreateAuthTables
        Running: (CodeIgniter\Settings) 2021-07-04-041948_CodeIgniter\Settings\Database\Migrations\CreateSettingsTable
        Running: (CodeIgniter\Settings) 2021-11-14-143905_CodeIgniter\Settings\Database\Migrations\AddContextColumn
Migrations complete.
```

- Shield uses Migrations to setup new database tables!
- This is why you were taught Migrations in Week 4.

CodeIgniter Shield Database Tables



Shield Config

app > Config > Routes.php

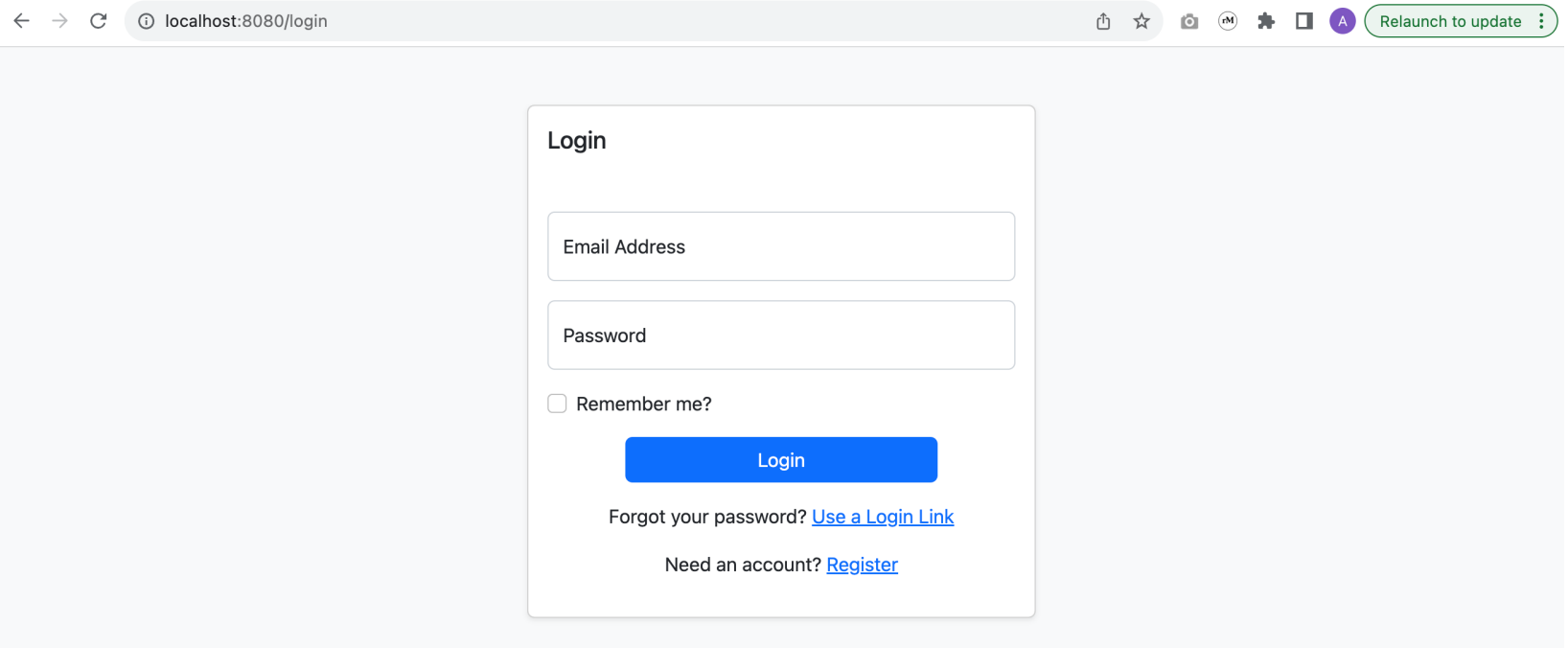
```php
<?php

use CodeIgniter\Router\RouteCollection;

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

service('auth')->routes($routes);

```

app > Config > Auth.php

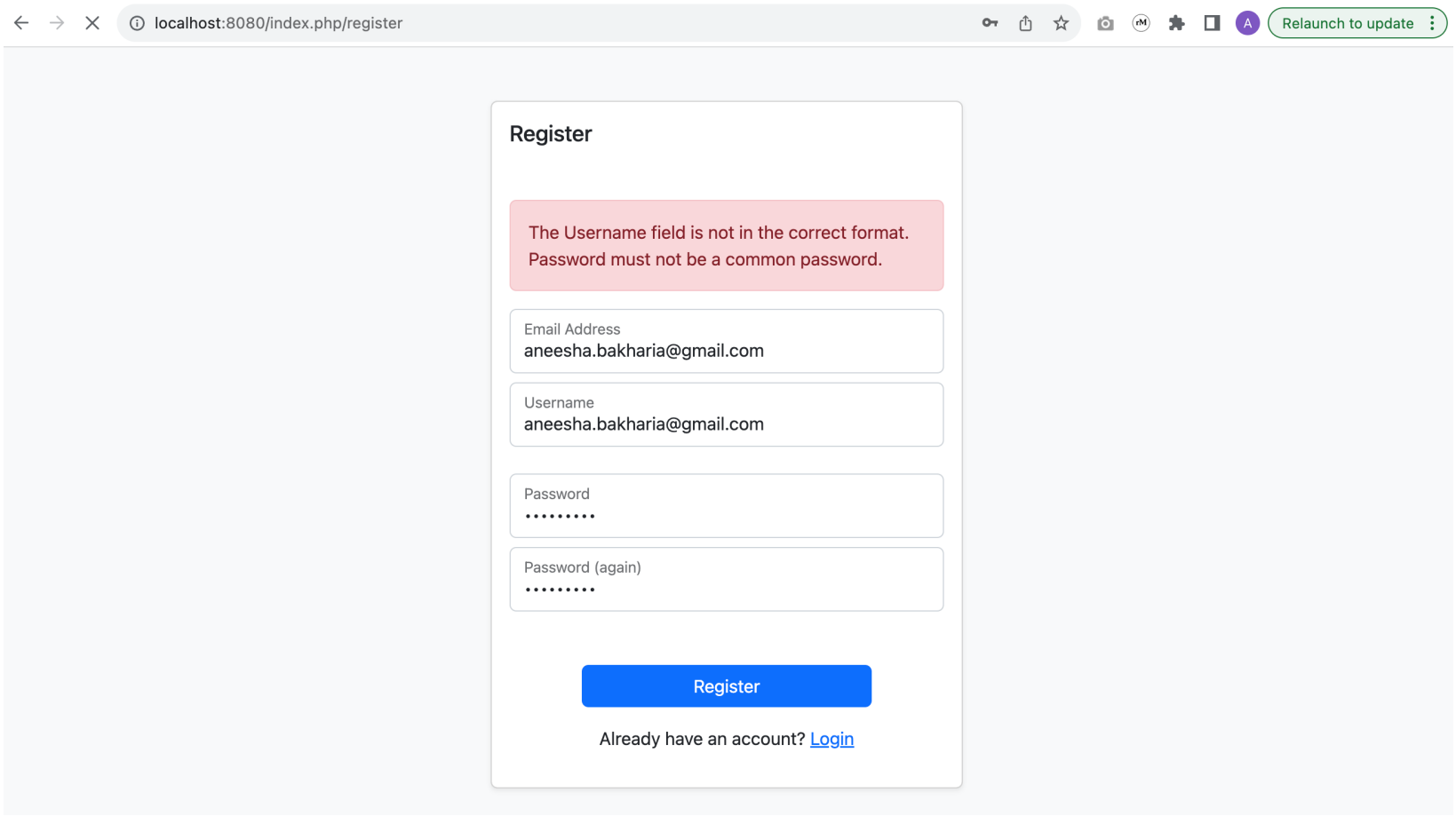
```php
{

    /**
     * -------------------------------------------------------------------
     * View files
     * -------------------------------------------------------------------
     */
    public array $views = [
        'login'                      => '\CodeIgniter\Shield\Views\login',
        'register'                   => '\CodeIgniter\Shield\Views\register',
        'layout'                     => '\CodeIgniter\Shield\Views\layout',
        'action_email_2fa'           => '\CodeIgniter\Shield\Views\email_2fa_show',
        'action_email_2fa_verify'    => '\CodeIgniter\Shield\Views\email_2fa_verify',
        'action_email_2fa_email'     => '\CodeIgniter\Shield\Views\Email\email_2fa_email',
        'action_email_activate_show' => '\CodeIgniter\Shield\Views\email_activate_show',
        'action_email_activate_email' => '\CodeIgniter\Shield\Views\Email\email_activate_email',
        'magic-link-login'           => '\CodeIgniter\Shield\Views\magic_link_form',
        'magic-link-message'         => '\CodeIgniter\Shield\Views\magic_link_message',
        'magic-link-email'           => '\CodeIgniter\Shield\Views\Email\magic_link_email',
    ];

    /**
     * -------------------------------------------------------------------
     * Redirect URLs
     * -------------------------------------------------------------------
     * The default URL that a user will be redirected to after various auth
     * actions. This can be either of the following:
     *
     * 1. An absolute URL. E.g. http://example.com OR https://example.com
     * 2. A named route that can be accessed using `route_to()` or `url_to()`
     * 3. A URI path within the application. e.g 'admin', 'login', 'expath'
     *
     * If you need more flexibility you can override the `getUrl()` method
     * to apply any logic you may need.
     */
    public array $redirects = [
        'register'         => '/',
        'login'            => '/',
        'logout'           => 'login',
        'force_reset'      => '/',
        'permission_denied' => '/',
        'group_denied'     => '/',
    ];
```

- You can clone the views and then add your own with a customised look and feel

Shield – Default Login View



Shield – Default Register View



Shield – Authgroups & Permissions

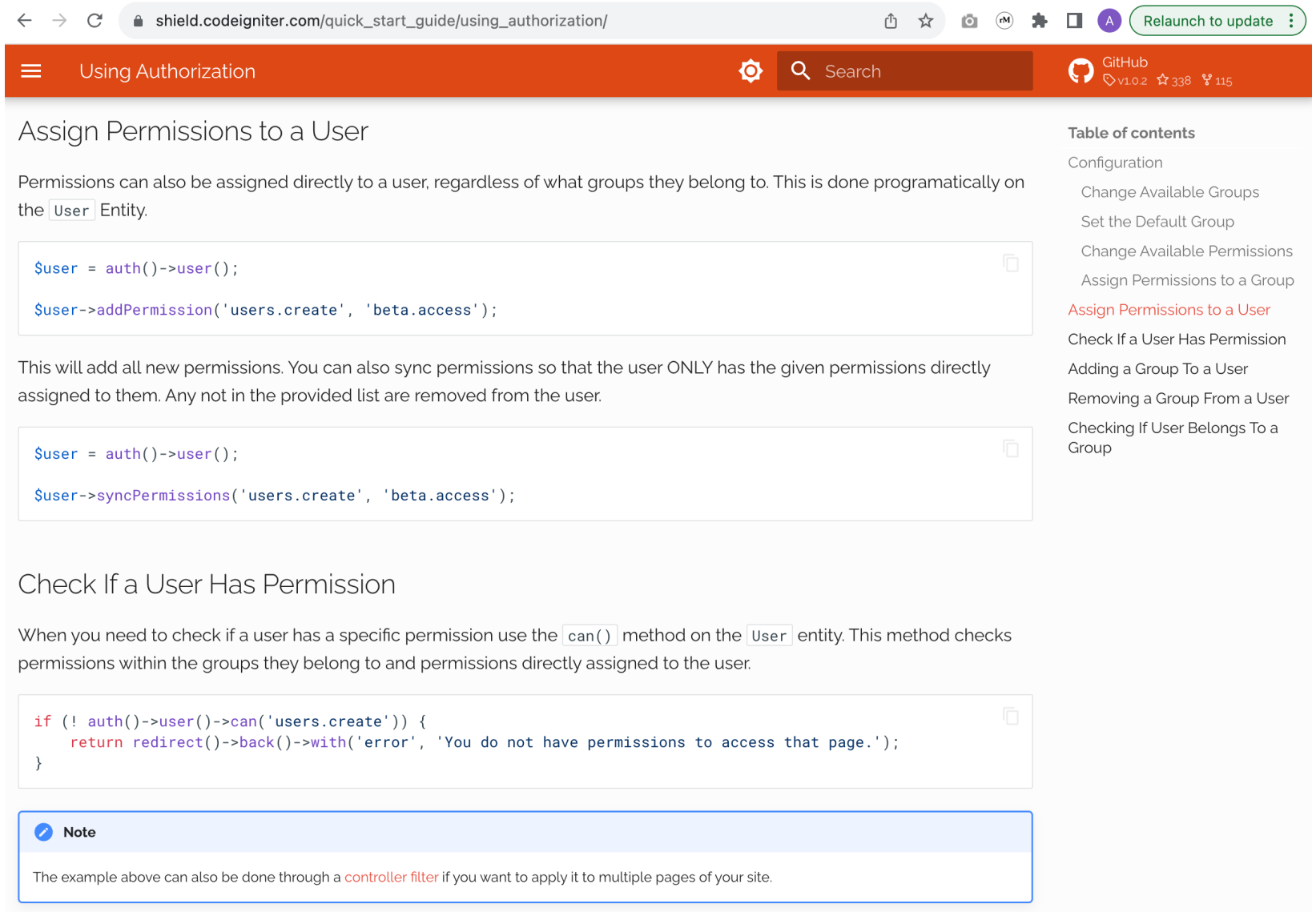
app > Config > AuthGroups.php

```
19  {
28      /**
29       * --------------------------------------------------------------------
30       * Groups
31       * --------------------------------------------------------------------
32       * An associative array of the available groups in the system, where the keys
33       * are the group names and the values are arrays of the group info.
34       *
35       * Whatever value you assign as the key will be used to refer to the group
36       * when using functions such as:
37       *      $user->addGroup('superadmin');
38       *
39       * @var array<string, array<string, string>>
40       *
41       * @see https://codeigniter4.github.io/shield/quick_start_guide/using_authorization/#change-available-groups for more info
42       */
43      public array $groups = [
44          'superadmin' => [
45              'title'       => 'Super Admin',
46              'description' => 'Complete control of the site.',
47          ],
48          'admin' => [
49              'title'       => 'Admin',
50              'description' => 'Day to day administrators of the site.',
51          ],
52          'developer' => [
53              'title'       => 'Developer',
54              'description' => 'Site programmers.',
55          ],
56          'user' => [
57              'title'       => 'User',
58              'description' => 'General users of the site. Often customers.',
59          ],
60          'beta' => [
61              'title'       => 'Beta User',
62              'description' => 'Has access to beta-level features.',
63          ],
64      ];
```

app > Config > AuthGroups.php

```
19  {
66      /**
67       * --------------------------------------------------------------------
68       * Permissions
69       * --------------------------------------------------------------------
70       * The available permissions in the system.
71       *
72       * If a permission is not listed here it cannot be used.
73       */
74      public array $permissions = [
75          'admin.access'        => 'Can access the sites admin area',
76          'admin.settings'      => 'Can access the main site settings',
77          'users.manage-admins' => 'Can manage other admins',
78          'users.create'        => 'Can create new non-admin users',
79          'users.edit'          => 'Can edit existing non-admin users',
80          'users.delete'        => 'Can delete existing non-admin users',
81          'beta.access'         => 'Can access beta-level features',
82      ];
83
84      /**
85       * --------------------------------------------------------------------
86       * Permissions Matrix
87       * --------------------------------------------------------------------
88       * Maps permissions to groups.
89       *
90       * This defines group-level permissions.
91       */
92      public array $matrix = [
93          'superadmin' => [
94              'admin.*',
95              'users.*',
96              'beta.*',
97          ],
98          'admin' => [
99              'admin.access',
100             'users.create',
101             'users.edit',
102             'users.delete',
103             'beta.access',
104         ],
105         'developer' => [
106             'admin.access',
107             'admin.settings',
108             'users.create',
109             'users.edit',
110             'beta.access',
111         ],
```

Shield – Setting and Checking Permissions



https://shield.codeigniter.com/quick_start_guide/using_authorization/

Shield – Getting Practise

- In Lab 7, you'll use Shield to add Auth and Permissions to the ResumeBuilder Application

- In the Final Project you could do one of the following:

  - create your own login and permissions
    (saving encrypted passwords in the database)

  - use the Shield library (if you are using CodeIgniter)

  - use Github or Google social login (but will need a developer token)

Week 6: Todo

- Weekly RiPPLE task is due in Week 6 on Thursday at 3pm
- Complete Week 6 Lab 5
  - You'll create a CRUD Admin Panel for the ResumeBuilder app
- Continue with your Design Document Assessment Item
  - You should now be finishing the design document

INFS3202/7202 – Our Progress

Learning Objectives

From Course Profile

1. Apply system architecture principles to design and deploy Web Information Systems (WIS) solutions.
2. Evaluate and articulate the scope, complexity, and key considerations in the design and implementation of Web Information Systems.
3. Design and program Web Information Systems (WIS) with server-side functionalities.
4. Develop responsive Web-based, database-driven applications using efficient and effective technologies.
5. Evaluate and justify the suitability of Web Information Systems solutions in various contexts, considering factors such as user needs and technical constraints.
6. Judge in which situations WIS solutions are more or less appropriate.
7. Critically analyze current issues and emerging trends in Web Information Systems development, and predict potential impacts on future practices and technologies.

Lectures and Practicals – Week 1 to 6

Week	Lecture	Practical
Week 1	Course Overview & Intro to WWW	No Practical in Week 1
Week 2	Creating and Deploying Web Applications (includes HTML, CSS Recap, PHP and basic UI layouts)	Practical 1: UQCloud, HTML and PHP
Week 3	MVC 1 – Controller and View (includes UX prototyping with CSS libraries)	Practical 2: Building your First CodeIgniter Project
Week 4	MVC 2 – Models & SQL Databases (includes database design patterns, SQL and Postgres database, and Faceted Search)	Practical 3: Databases and Models
Week 5	MVC 3 – Creating CRUD Applications	Practical 4: Designing UI's with CSS Frameworks
Week 6	MVC 4 – Advanced topics (File uploads, caching, sessions, authentication & authorisation)	Practical 5: Search and Form Processing
Mid-Semester Break		

Lectures and Practicals – Week 7 to 13

Week	Lecture	Practical
Week 7	Incorporating GenAI features in Web Applications (Calling GenAI API's, Creating Chatbots and Retrieval Augmented Generation)	Practical 6: Incorporating GenAI
Week 8	Designing RESTFul API's & JavaScript (CodeIgniter RESTful resource handling, API Auth tokens)	Practical 7: Creating RESTFul API's
Week 9	Developing Progressive Web Applications (Responsive CSS, PWA's, Accessibility & Multiple Language Support)	Practical 8: Work on Project
Week 10	Deploying to the Cloud - Guest Lecturer from AWS Solution Architects (VPC, Route66, EC2, Gateway and Load Balancing, Container Deployment, Serverless)	Project Code Review
Week 11	Web Security & Guest Lecture – Working as a Web Developer	Practical 9: Deployment to AWS
Week 12	Other MVC frameworks (Flask, Django, FastAPI, Next.JS) & NoSQL Databases	Practical 10: Work on Project
Week 13	Revision	Practical 11: Exam Revision

By the end of the course you should be able to go from an idea to a web application:

- Design a database
- Design and Implement the UI
- Program the functionality in a server-side programming language
  - Still need to cover testing, logging and API’s
- Deploy the application (in the cloud e.g. AWS)
- Use GenAI Ethically and Creatively

Thank you