**Menu Master Program Tests**

Discussion of the development and testing aspects.

**Development:**

**Development Environment and Tools:**

1. **Framework:** Laravel (Backend):

I used a PHP framework called Laravel to implement this application. It follows the Model-View-Controller (MVC) architecture and has features and libraries for route, authentication, and database management.

1. **Frontend:** I used HTML, CSS, and JavaScript for the user interface:

HTML (Hypertext Markup Language) to structure the contents of the website pages, CSS (Cascading Style Sheets) to style and layout the web contents, and JavaScript to add dynamic behaviour and interactivity to website contents.

1. **Database:** This application uses MySQL for database development and management of data related to menu items, messages, feedback, and users.
2. **IDE:** This application was developed using Visual Studio Code.

**Major Components or Modules:**

1. **User Management**: The system allows admin and restaurant owners are able to register and log in.
   1. Login request with email -> [admin@admin.com](mailto:admin@admin.com) is recognized as super admin and is logged in to the admin dashboard while the rest are logged in to the business owner’s dashboard.
2. **Menu Items Management:** Only the super admin and the specific business owners that added the menu items are able to perform operations on them such as editing and deleting the menu items.
3. **Message System:** the system allows users to send messages to different restaurant owners to which the restaurant owners can reply privately i.e. maybe through emails.
4. **Feedback System:** The system Collects and displays user feedback to the admin.
5. **Authentication and Authorization**: on registration, users are authenticated through a password system that requires them to enter a password that has the following criteria:
   * 1. Minimum of 8 characters.
     2. At least one uppercase letter
     3. At least one lowercase letter
     4. At least one special Character
     5. At least one Number.
   * The system also implements role-based access control i.e., admin or restaurant owner.

**Challenges Faced During Implementation:**

* **Internet connection Issues**: most of the Laravel dependencies and other third-party libraries such as composer, node.js, etc., require a stable internet connection for efficient installation.
* **Complex Relationships:** creating and managing the complex relationships between entities and ensuring data integrity was a challenge.
* **Security concerns:** it was a challenging task to ensure data security, prevent SQL injection, and handle user authentication securely which required close intervention.
* **Scalability:** Ensuring the system can handle a growing number of menu items, and maybe other business owners when required.
* **Version concerns:** tools and some other third-party libraries require the latest versions for others to be compatible such as PHP, composer, and node.js.

**Third-Party Libraries or Frameworks:**

1. **Node.js** : I used this library to manage frontend assets, build processes, and JavaScript-related tasks to enhance the user interface of the application such “npm run dev”.
2. **Composer** : this library was essential for managing PHP dependencies, including Laravel and other PHP packages, to ensure smooth development and deployment of the backend part of the application such as “php artisan”.
3. **Bootstrap:** For laying out and designing the frontend contents.
4. **PHPUnit**: For integration and unit testing.

**Testing:**

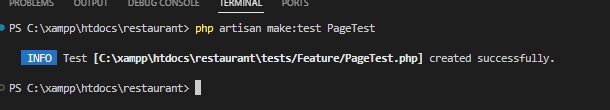
**Testing Strategy:**

* **Unit Testing:** this was achieved by using the inbuild Laravel Unit Test functionality for testing individual functions, methods, and components in isolation such as test for web pages and responses.
* **Integration Testing:** this was achieved by using the inbuild Laravel Feature Test functionality for testing the interaction between various modules and components such as authentication, authorization, HTTP tests e.g., POST, DELETE, GET, and also database test such as checking if the user exists in the database by validating the user emails.
* **Acceptance Testing:** this was achieved by testing the application as a whole, including user scenarios and workflow.

**Test Cases and Results:**

* **Performed Feature and Unit Tests in Laravel**

**Making the Feature test**



**Writing the tests**

**Pages tests**

<?php

namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;

use Illuminate\Foundation\Testing\WithFaker;

use Tests\TestCase;

class PageTest extends TestCase

{

    /\*\*

     \* A basic feature test example.

     \*

     \* @return void

     \*/

    public function test\_homepage()

    {

        $response = $this->get('/');

        $response->assertStatus(200);

    }

    public function test\_courtsidepage()

    {

        $response = $this->get('/courtside');

        $response->assertStatus(200);

    }

    public function test\_beachsidepage()

    {

        $response = $this->get('/beachside');

        $response->assertStatus(200);

    }

    public function test\_riversidepage()

    {

        $response = $this->get('/riverside');

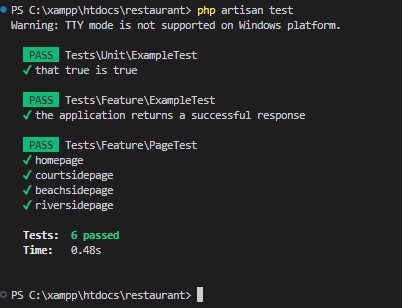
        $response->assertStatus(200);

    }

}

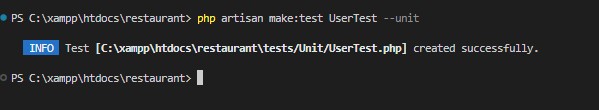
**Run the Command in the terminal**:

-> php artisan test



**Unit Testing**

**Making Unit test**



**Testing the Login and Register form in Unit tests**

**Login and Register Form test**

<?php

namespace Tests\Unit;

use Tests\TestCase;

class UserTest extends TestCase

{

    /\*\*

     \* A basic unit test example.

     \*

     \* @return void

     \*/

    public function test\_login\_form()

    {

        $response = $this->get('/login');

        $response->assertStatus(200);

    }

public function test\_register\_form()

    {

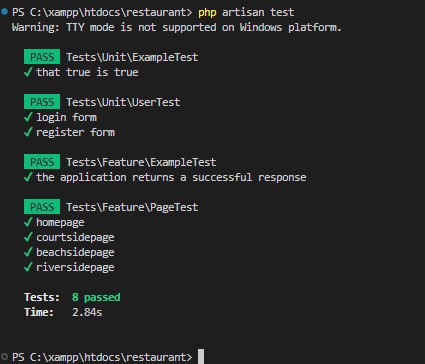
        $response = $this->get('/register');

        $response->assertStatus(200);

    }

}

**Login and Register forms test result**



**Testing for User Duplication**

<?php

namespace Tests\Unit;

use App\Models\User;

use Tests\TestCase;

class UserTest extends TestCase

{

    /\*\*

     \* A basic unit test example.

     \*

     \* @return void

     \*/

    public function test\_login\_form()

    {

        $response = $this->get('/login');

        $response->assertStatus(200);

    }

    public function test\_register\_form()

    {

        $response = $this->get('/register');

        $response->assertStatus(200);

    }

    public function test\_user\_duplication(){

        $user1 = User::make([

            'name' => 'johnD',

            'email' => 'johnd@gmail.com'

        ]);

        $user2 = User::make([

            'name' => 'danc',

            'email' => 'danc@gmail.com'

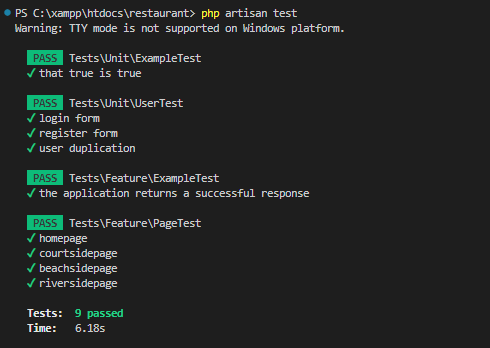
        ]);

        $this->assertTrue($user1->email != $user2->email);

    }

}

**It is a PASS test if the user emails are not the same**



**Testing with the same user emails**

<?php

namespace Tests\Unit;

use App\Models\User;

use Tests\TestCase;

class UserTest extends TestCase

{

    /\*\*

     \* A basic unit test example.

     \*

     \* @return void

     \*/

    public function test\_login\_form()

    {

        $response = $this->get('/login');

        $response->assertStatus(200);

    }

    public function test\_register\_form()

    {

        $response = $this->get('/register');

        $response->assertStatus(200);

    }

    public function test\_user\_duplication(){

        $user1 = User::make([

            'name' => 'johnD',

            'email' => 'johnd@gmail.com'

        ]);

        $user2 = User::make([

            'name' => 'danc',

            'email' => 'johnd@gmail.com'

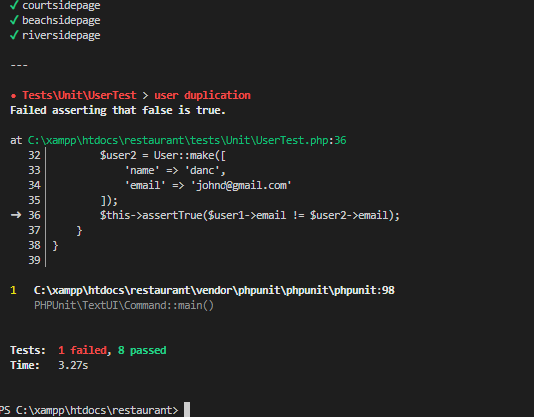
        ]);

        $this->assertTrue($user1->email != $user2->email);

    }

}

**It will fail the test if the user emails are the same**



**Delete User Test**

<?php

namespace Tests\Unit;

use App\Models\User;

use Tests\TestCase;

class UserTest extends TestCase

{

    /\*\*

     \* A basic unit test example.

     \*

     \* @return void

     \*/

    public function test\_login\_form()

    {

        $response = $this->get('/login');

        $response->assertStatus(200);

    }

    public function test\_register\_form()

    {

        $response = $this->get('/register');

        $response->assertStatus(200);

    }

//user duplication test

    public function test\_user\_duplication(){

        $user1 = User::make([

            'name' => 'johnD',

            'email' => 'johnd@gmail.com'

        ]);

        $user2 = User::make([

            'name' => 'danc',

            'email' => 'danc@gmail.com'

        ]);

        $this->assertTrue($user1->email != $user2->email);

    }

//delete user test

    public function test\_delete\_user(){

        $user =User::factory()->count(1)->make();

        $user = User::first();

        if($user){

            $user->delete();

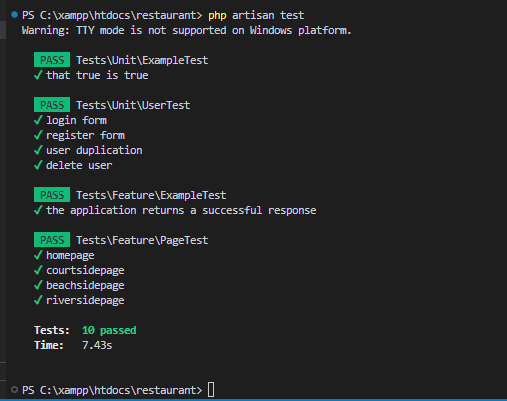
        }

        $this->assertTrue(true);

    }

}

**If the User exists if the record in Count() it will pass the delete test**



**HTTP Tests : POST**

**Creating new User Test**

<?php

namespace Tests\Unit;

use App\Models\User;

use Tests\TestCase;

class UserTest extends TestCase

{

    /\*\*

     \* A basic unit test example.

     \*

     \* @return void

     \*/

    public function test\_login\_form()

    {

        $response = $this->get('/login');

        $response->assertStatus(200);

    }

    public function test\_register\_form()

    {

        $response = $this->get('/register');

        $response->assertStatus(200);

    }

    public function test\_user\_duplication(){

        $user1 = User::make([

            'name' => 'johnD',

            'email' => 'johnd@gmail.com'

        ]);

        $user2 = User::make([

            'name' => 'danc',

            'email' => 'danc@gmail.com'

        ]);

        $this->assertTrue($user1->email != $user2->email);

    }

    public function test\_delete\_user(){

        $user =User::factory()->count(1)->make();

        $user = User::first();

        if($user){

            $user->delete();

        }

        $this->assertTrue(true);

    }

    // test to store new user on registration

    public function test\_it\_stores\_new\_user(){

        $response = $this->post('register',[

            'name'=> 'dmax',

            'email'=>'dmax@gmail.com',

            'phone\_number'=>'1234567890',

            'access'=>'beachside',

            'password'=>'12345678',

            'password\_confirmation'=>'12345678',

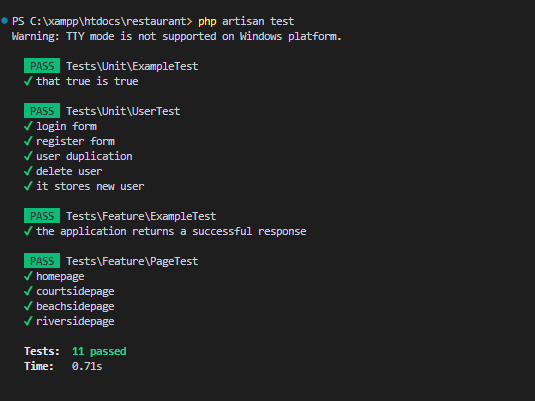
        ]);

        $response->assertRedirect('/home');

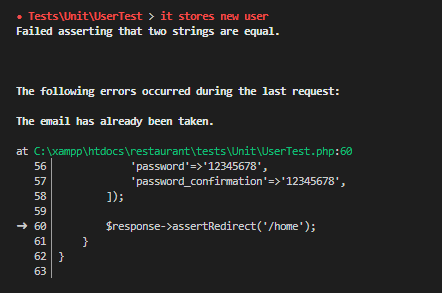
    }

}

**Create new user test result pass**



***It Fails if the email has already been taken***



**Send Message Tests**

<?php

namespace Tests\Unit;

use Tests\TestCase;

class MessageTest extends TestCase

{

    /\*\*

     \* A basic unit test example.

     \*

     \* @return void

     \*/

    public function test\_message\_beachside(){

        $response = $this->post('beachside',[

             'name'=> 'dmax',

             'email'=>'aggie@gmail.com',

             'subject'=>'nice',

             'message'=>'nice',

             'access'=>'beachside',

         ]);

         $response->assertRedirect('/beachside');

     }

     public function test\_message\_riverside(){

        $response = $this->post('riverside',[

             'name'=> 'dmax',

             'email'=>'aggie@gmail.com',

             'subject'=>'nice',

             'message'=>'nice',

             'access'=>'riverside',

         ]);

         $response->assertRedirect('/riverside');

     }

     public function test\_message\_courtside(){

        $response = $this->post('courtside',[

             'name'=> 'dmax',

             'email'=>'aggie@gmail.com',

             'subject'=>'nice',

             'message'=>'nice',

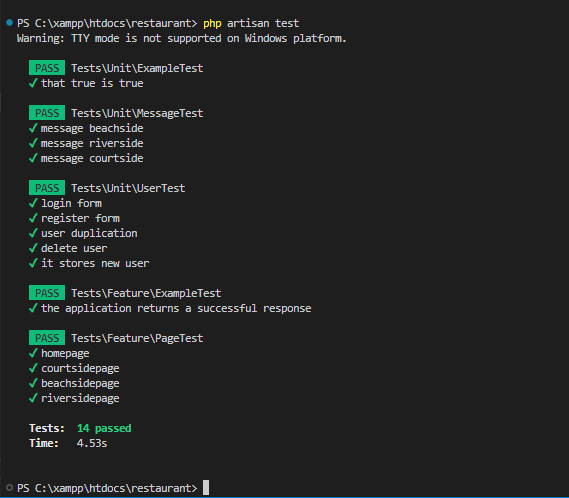
             'access'=>'courtside',

         ]);

         $response->assertRedirect('/courtside');

     }

}



**Test Cases and Results Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **TEST CASE** | **EXPECTED RESULT** | **ACTUAL RESULTS** | **TEST PASS/FAIL** |
| test\_homepage() | HTTP status code 200 (OK) | HTTP status code 200 | Pass |
| test\_courtsidepage() | HTTP status code 200 (OK) | HTTP status code 200 | Pass |
| test\_beachsidepage() | HTTP status code 200 (OK) | HTTP status code 200 | Pass |
| test\_riversidepage() | HTTP status code 200 (OK) | HTTP status code 200 | Pass |
| test\_login\_form() | HTTP status code 200 (OK) | HTTP status code 200 | Pass |
| test\_register\_form() | HTTP status code 200 (OK) | HTTP status code 200 | Pass |
| test\_user\_duplication()  (**WITH DIFFERENT EMAILS**) | Assert True is True (OK)  The test should pass Since $user1 and $user2 have different email addresses | Success asserting that fail is True | Pass |
| test\_user\_duplication()  **WITH SAME EMAILS**) | Assert True is True (OK)  The test should fail Since $user1 and $user2 have the same email addresses | Failed asserting that fail is True | Fail |
| test\_delete\_user() | Assert True is True (OK)  The test is expected to pass as long as the user is deleted without any errors because of the assertTrue(true) | Success asserting that fail is True  user is deleted without errors | Pass |
| test\_it\_stores\_new\_user() | successfully register, and after registration, the user is redirected to the '/home' URL. | successful and the redirection to '/home' occurs. | Pass |
| test\_it\_stores\_new\_user()  (**with email already existing)** | successfully register with no errors, and after registration, the  user is redirected to the '/home' URL. | Email has already been taken | Fail |
| test\_message\_beachside() | Pass if a message is successfully posted for the 'beachside' access location, and after posting, the user is redirected to the '/beachside' URL | The message is posted successfully and the redirection to '/beachside' occurs | Pass |
| test\_message\_riverside() | Pass if a message is successfully posted for the 'riverside' access location, and after posting, the user is redirected to the '/riverside' URL. | The message is posted successfully and the redirection to '/riverside' occurs. | Pass |
| test\_message\_courtside() | Pass if a message is successfully posted for the 'courtside' access location, and after posting, the user is redirected to the '/courtside' URL. | The message is posted successfully and the redirection to '/courtside' occurs. | Pass |

**Issue and Bug Management:**

* **Version and Milestone Tagging:** Software tools and libraries keep changing hence all the libraries' versions have been tagged in the “composer.json” file in the Menu Master folder to track issues with the specific software version or milestone they relate to. This is to help in planning and ensuring that certain issues are addressed in a particular release.
* **Performing regular updates** to the issue tracking system and system components to provide compatibility with the released libraries and tools.
* **Feedback sections** have been implemented in the home and restaurant web pages so that users can report any issues and bugs they find challenging and suggest improvements.
* **Bug management strategies –** providing bug identification mechanisms such as testing, and user feedback. Investigating the root causes of the bugs then performing code analysis, and database inspection to understand why the issues occurred. Fixing the bugs by making necessary code and database changes and ensuring that they align with the application’s coding standards. Then testing and verifying if the bugs have been fixed and then documenting the changes that have been made.

**Code Coverage and Quality:**

* High code coverage has been ensured in this system, especially for the critical parts of the system such as the authentication and authorization features.
* This system code coverage has been measured using a tool called PHPUnit code coverage report.
* Code reviews and automated quality checks were part of the development process which facilitated the maintenance of high code quality.

**Entity Relational Diagram**

MENU\_ITEM

USER

MESSAGE

FEEDBACK

1

**receives**

**Create &manages**

1

1

M

M

M