Simple CRUD with Codeigniter and MySQL

Name: Rhazzhamir I. Aguilar ***BSIT-3B***

Barangay Information and Management System (BIMS)

What is CRUD?

CRUD stands for:

- **C - Create (Add new data)**
- **R - Read (Display or view data)**
- **U - Update (Edit or modify data)**
- **D - Delete (Remove data)**

In this tutorial, we will build a simple student list web app that allows you to Create, Read, and Delete student information using CodeIgniter 4.

What is CodeIgniter?

CodeIgniter is a lightweight PHP framework used to build web applications quickly and easily. It's beginner-friendly and works well for simple systems like inventory, student records, etc.

🔁 **Recap of Activity 5**

Before continuing with this tutorial (Activity 6), let's quickly review what was done in **Activity 5**:

- ✅ **Step 1**: Searched and explored the official **CodeIgniter Documentation**
- ✅ **Step 2**: Installed **Composer** as a PHP dependency manager
- ✅ **Step 3**: Created a new **CodeIgniter Project** using Composer
- ✅ **Step 4**: Built a simple **User Interface (UI)** layout for the CRUD system
- ✅ **Step 5**: Set up a **Controller** to manage logic and flow
- ✅ **Step 6**: Configured the **Routes** to connect URLs to controller actions
- ✅ **Step 7**:Creating User Dashboard

You should now have a basic CodeIgniter setup with UI, controller, and routing. If not, review Activity 5 before starting this tutorial.

🚀 Let's begin with **Activity 6**: Creating CRUD Operations

Step 1: Install Requirements

You'll need:

- A computer or laptop
- **XAMPP** (includes Apache + MySQL)

- **CodeIgniter 4**
- A browser (like Chrome)

Step 2: Open XAMPP and Start Services

- Open **XAMPP Control Panel**
- Start both **Apache** and **MySQL**
- Open vscode

Step 3: Connect CodeIgniter to the Database

1. Inside your CodeIgniter project folder, look for a file named env

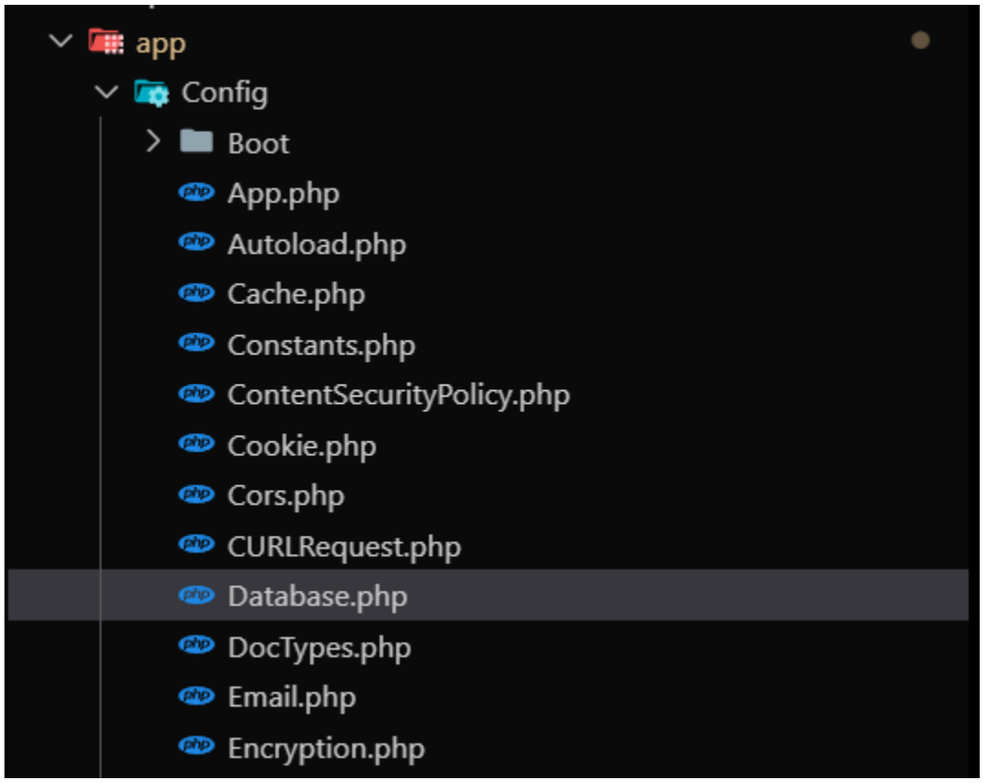
2. **Copy** it, **paste** it in the same folder, and rename it to .env

3. Open .env and **uncomment and edit** the following settings:

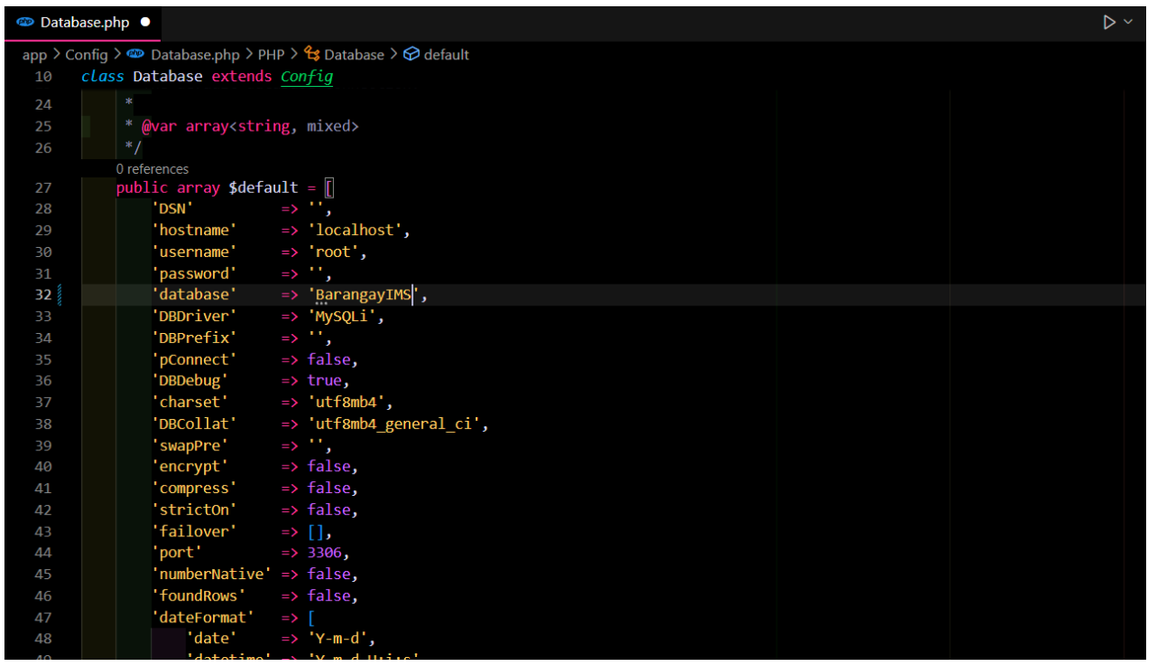
```
.env
15  #--------------------------------------------------------------------
16
17  CI_ENVIRONMENT = development
18
19  #--------------------------------------------------------------------
20  # APP
21  #--------------------------------------------------------------------
22
23  # app.baseURL = ''
24  # If you have trouble with `.`, you could also use `_`.
25  # app_baseURL = ''
26  # app.forceGlobalSecureRequests = false
27  # app.CSPEnabled = false
28
29  #--------------------------------------------------------------------
30  # DATABASE
31  #--------------------------------------------------------------------
32
33  database.default.hostname = localhost
34  database.default.database = BarangayIMS
35  database.default.username = root
36  database.default.password =
37  database.default.DBDriver = MySQLi
38  database.default.DBPrefix =
39  database.default.port = 3306
40
```

Alternatively, you can also configure the database in the app/Config/Database.php file:

1. Open the file app/Config/Database.php.



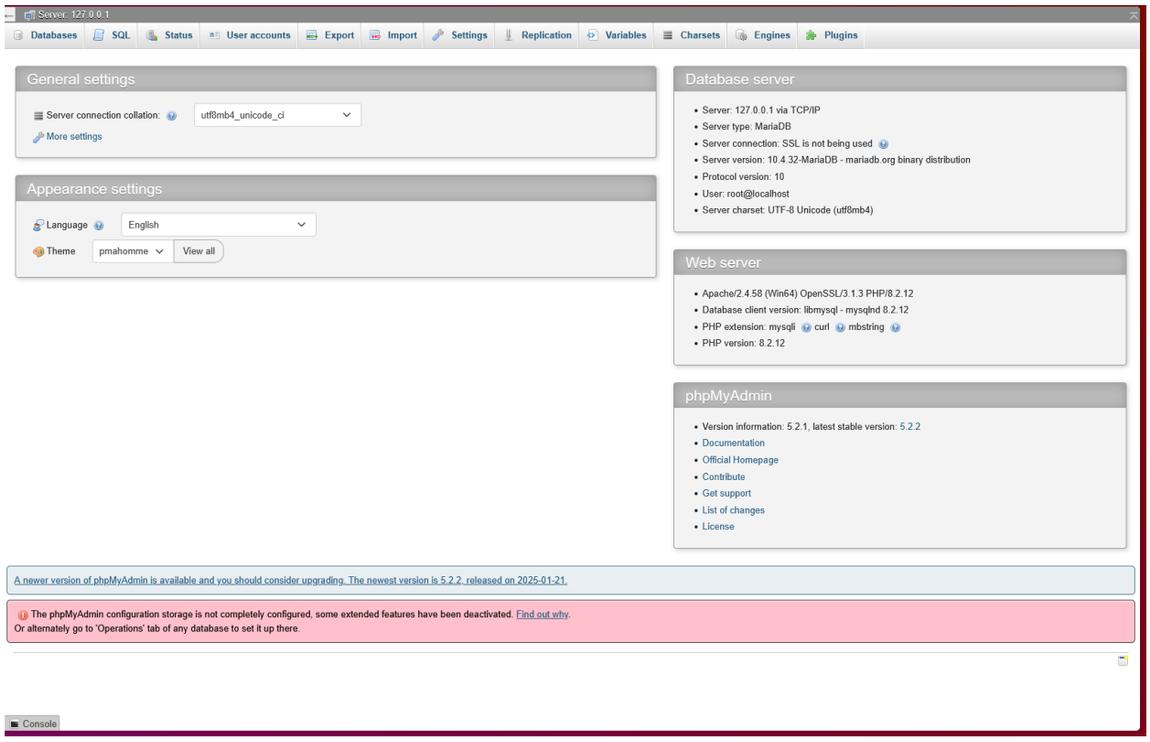
2. Look for the $default array and edit it like this:



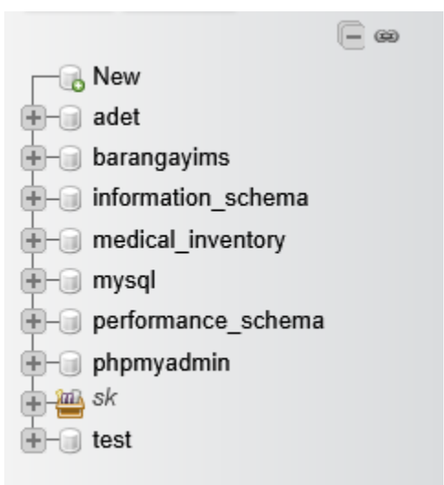
Step 5: Create the Database and Table

1. Create the Database:

- Go to http://localhost/phpmyadmin (make sure XAMPP is running).



- Click on New to create a database.

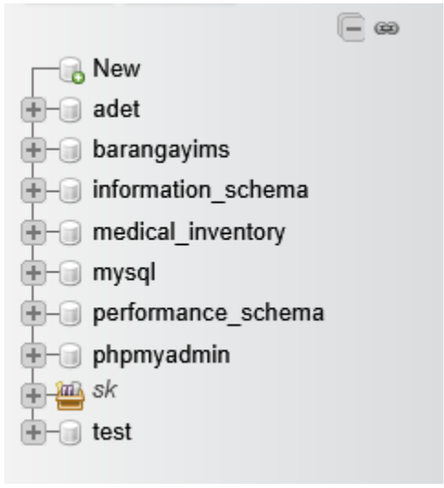


- Name the database BarangayIMS (or whatever name you chose in the .env file) and click Create.

2. *Create a Table:*

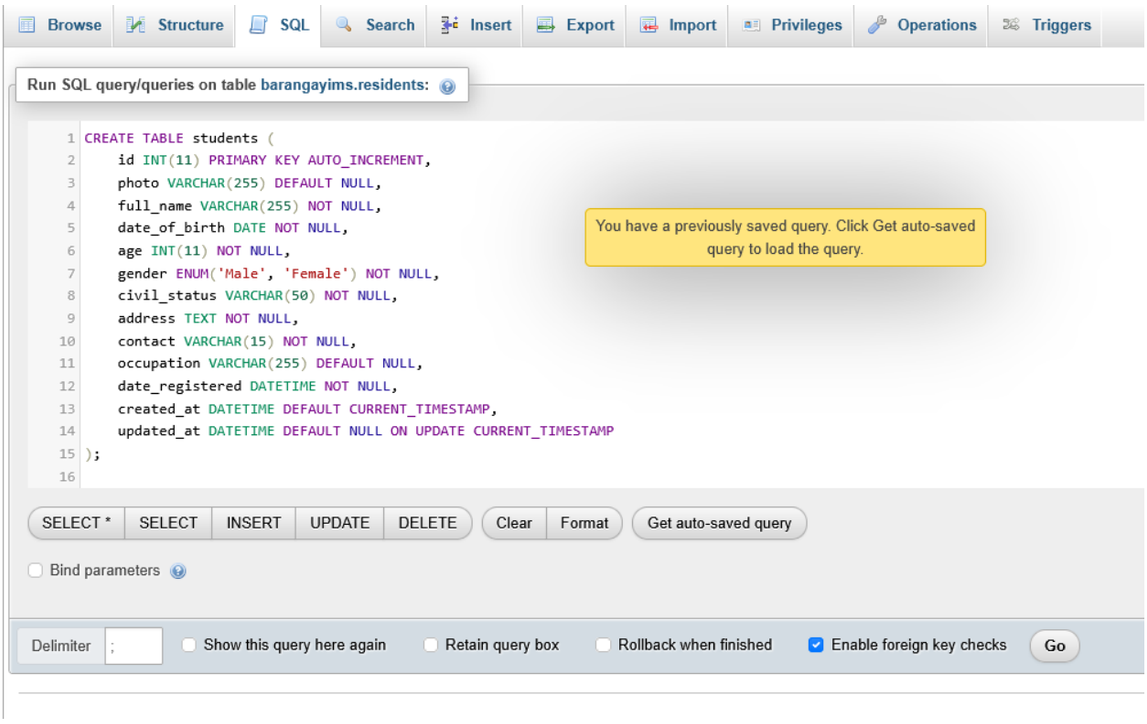
After creating the database, you'll need to create a table for storing student data. Follow these steps:

- In phpMyAdmin, select the BarangayIMS database.



- Click on the SQL tab at the top to run an SQL query.

- Paste the following SQL code to create a residents table:



Now that the **database** and **table** are set up, you can move on to **creating models**, **controllers**, ***routes*** and **views** for the CRUD operations.

Step 5 : Create a Model in CodeIgniter

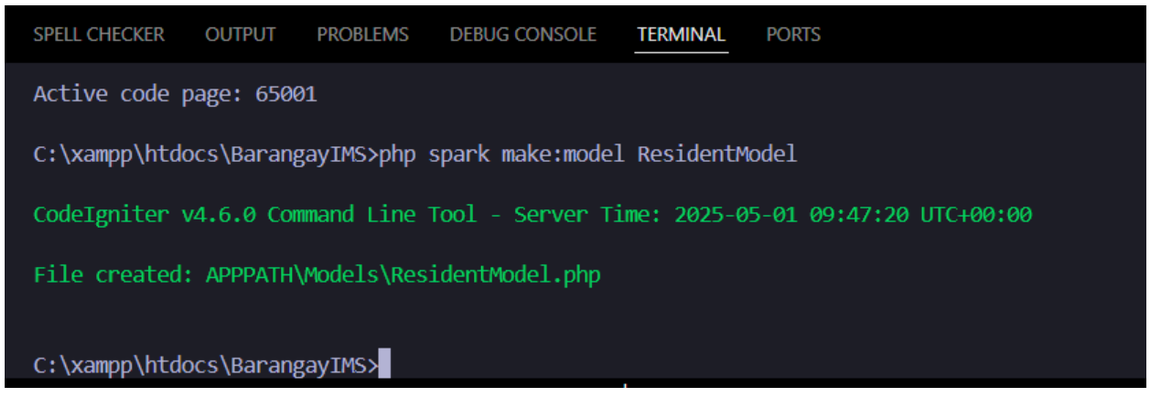
What is a Model?

A **Model** in CodeIgniter is a file that connects your application to the **database**.

- It is responsible for **retrieving**, **inserting**, **updating**, and **deleting** data.
- Think of it as the **bridge** between your database and your website.
- Models help keep your code organized by separating the database logic from the rest of your application (like the user interface and page structure).

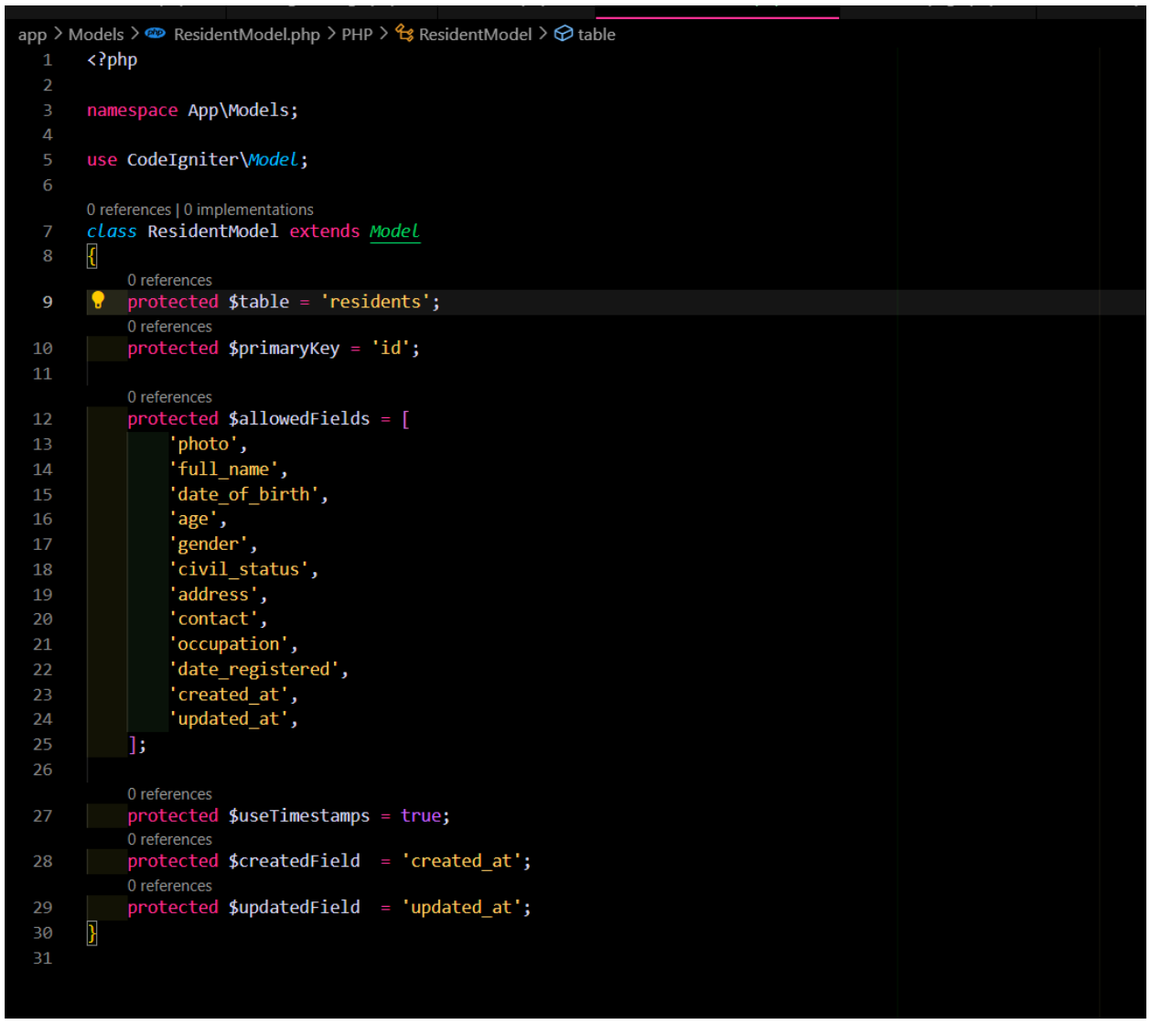
Simple Example: If you want to **add a student** to your system, the model is the part that actually saves that student's info into the database.

1. Open your terminal in the project root



Use this command to generate a model:

Here’s how your ResidentModel should look based on your table structure:



Notes: ⚠ Make sure the value of $table matches your actual table name in the database (e.g., students or residents).

Step 6: Creating a Controller (Part of Activity 5)

*note: Creating a controller was already introduced in **Activity 5**, but here we will go deeper as part of our full CRUD setup.*

What is a Controller?

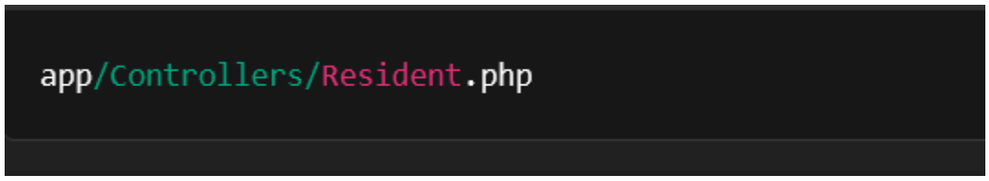
A **Controller** is the part of CodeIgniter that **handles user requests** and **controls what to do with them**.

- It acts like a traffic manager: when the user clicks a button or visits a page, the controller decides **which data to get**, **which page to show**, and **what action to perform** (like add, edit, or delete).
- It **talks to the Model** for data and **shows it using the View**.

How to Create a Controller

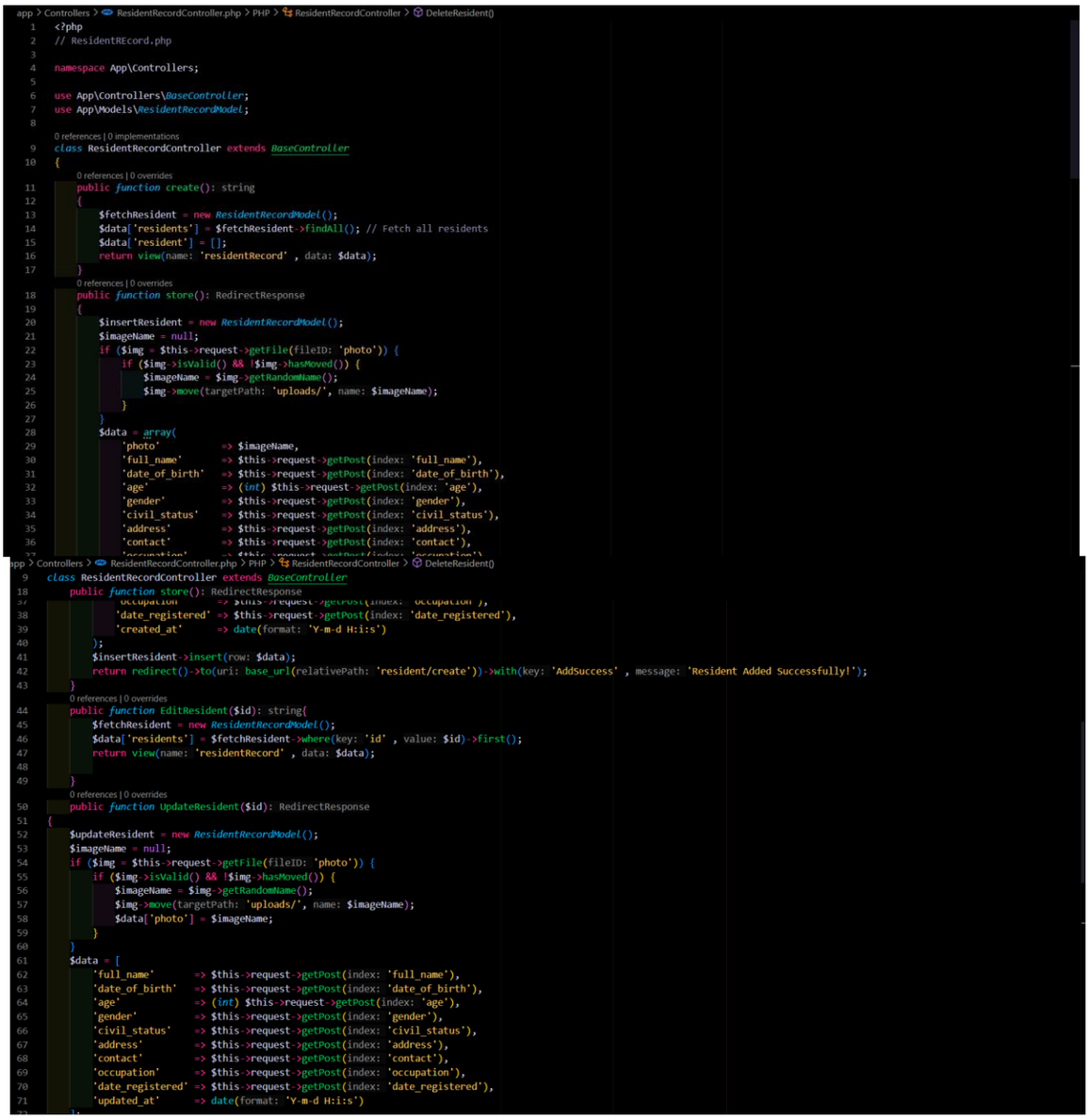
Use the terminal in your project directory and run:

This will create a file:



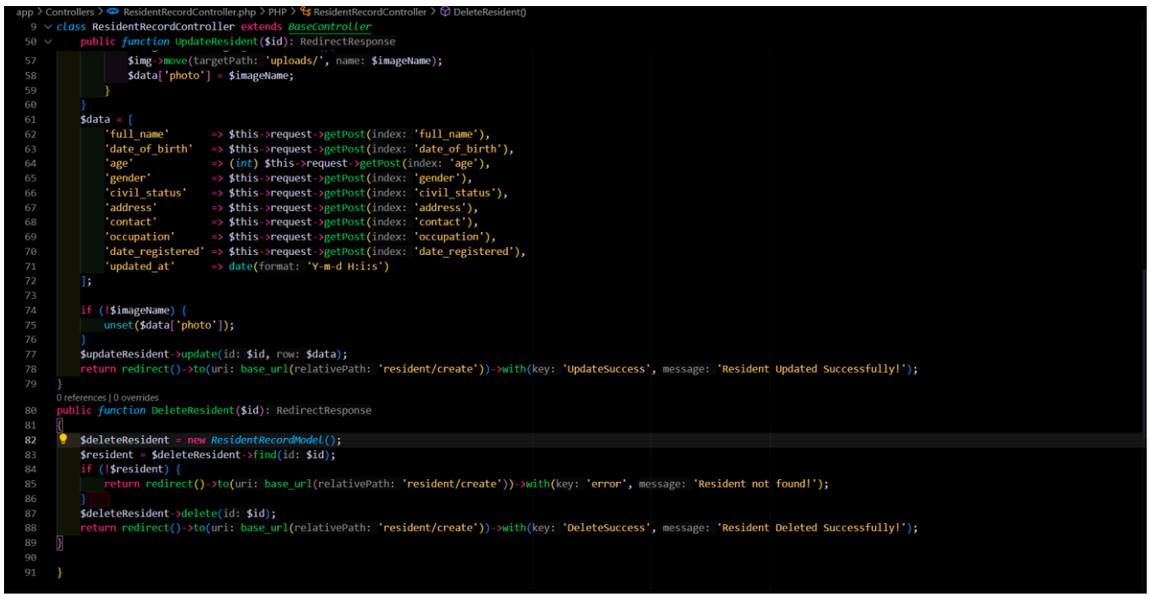
Edit the Controller

Open the Resident.php file and start with this basic setup:



```php
<?php
// ResidentREcord.php

namespace App\Controllers;

use App\Controllers\BaseController;
use App\Models\ResidentRecordModel;

class ResidentRecordController extends BaseController
{
    public function create(): string
    {
        $fetchResident = new ResidentRecordModel();
        $data['residents'] = $fetchResident->findAll(); // Fetch all residents
        $data['resident'] = [];
        return view('residentRecord' , $data);
    }
    public function store(): RedirectResponse
    {
        $insertResident = new ResidentRecordModel();
        $imageName = null;
        if ($img = $this->request->getFile('photo')) {
            if ($img->isValid() && !$img->hasMoved()) {
                $imageName = $img->getRandomName();
                $img->move('uploads/', $imageName);
            }
        }
        $data = array(
            'photo'          => $imageName,
            'full_name'      => $this->request->getPost('full_name'),
            'date_of_birth'  => $this->request->getPost('date_of_birth'),
            'age'            => (int) $this->request->getPost('age'),
            'gender'         => $this->request->getPost('gender'),
            'civil_status'   => $this->request->getPost('civil_status'),
            'address'        => $this->request->getPost('address'),
            'contact'        => $this->request->getPost('contact'),
```

```php
            'date_registered' => $this->request->getPost('date_registered'),
            'created_at'      => date('Y-m-d H:i:s')
        );
        $insertResident->insert($data);
        return redirect()->to(base_url('resident/create'))->with('AddSuccess' , 'Resident Added Successfully!');
    }
    public function EditResident($id): string{
        $fetchResident = new ResidentRecordModel();
        $data['residents'] = $fetchResident->where('id' , $id)->first();
        return view('residentRecord' , $data);

    }
    public function UpdateResident($id): RedirectResponse
{
    $updateResident = new ResidentRecordModel();
    $imageName = null;
    if ($img = $this->request->getFile('photo')) {
        if ($img->isValid() && !$img->hasMoved()) {
            $imageName = $img->getRandomName();
            $img->move('uploads/', $imageName);
            $data['photo'] = $imageName;
        }
    }
    $data = [
        'full_name'      => $this->request->getPost('full_name'),
        'date_of_birth'  => $this->request->getPost('date_of_birth'),
        'age'            => (int) $this->request->getPost('age'),
        'gender'         => $this->request->getPost('gender'),
        'civil_status'   => $this->request->getPost('civil_status'),
        'address'        => $this->request->getPost('address'),
        'contact'        => $this->request->getPost('contact'),
        'occupation'     => $this->request->getPost('occupation'),
        'date_registered' => $this->request->getPost('date_registered'),
        'updated_at'     => date('Y-m-d H:i:s')
```



```
app > Controllers > ResidentRecordController.php > PHP > ResidentRecordController > DeleteResident()
 9  class ResidentRecordController extends BaseController
50      public function UpdateResident($id): RedirectResponse
57              $img->move(targetPath: 'uploads/', name: $imageName);
58              $data['photo'] = $imageName;
59          }
60      }
61      $data = [
62          'full_name'       => $this->request->getPost(index: 'full_name'),
63          'date_of_birth'   => $this->request->getPost(index: 'date_of_birth'),
64          'age'             => (int) $this->request->getPost(index: 'age'),
65          'gender'          => $this->request->getPost(index: 'gender'),
66          'civil_status'    => $this->request->getPost(index: 'civil_status'),
67          'address'         => $this->request->getPost(index: 'address'),
68          'contact'         => $this->request->getPost(index: 'contact'),
69          'occupation'      => $this->request->getPost(index: 'occupation'),
70          'date_registered' => $this->request->getPost(index: 'date_registered'),
71          'updated_at'      => date(format: 'Y-m-d H:i:s')
72      ];
73
74      if (!$imageName) {
75          unset($data['photo']);
76      }
77      $updateResident->update(id: $id, row: $data);
78      return redirect()->to(uri: base_url(relativePath: 'resident/create'))->with(key: 'UpdateSuccess', message: 'Resident Updated Successfully!');
79  }
    0 references | 0 overrides
80  public function DeleteResident($id): RedirectResponse
81  {
82      $deleteResident = new ResidentRecordModel();
83      $resident = $deleteResident->find(id: $id);
84      if (!$resident) {
85          return redirect()->to(uri: base_url(relativePath: 'resident/create'))->with(key: 'error', message: 'Resident not found!');
86      }
87      $deleteResident->delete(id: $id);
88      return redirect()->to(uri: base_url(relativePath: 'resident/create'))->with(key: 'DeleteSuccess', message: 'Resident Deleted Successfully!');
89  }
90
91  }
```

What this Controller does:

- index() – show all residents
- create() – show form to add a resident
- store() – save new resident to database
- edit() – show form to edit an existing resident
- update() – update resident info
- delete() – delete a resident

Controller is now ready!

Step 7: Setting Up Routes (Part of Activity 5)

*note: Setting up routes was already introduced in **Activity 5**, but here we'll apply it in our actual CRUD system.*

What is a Route?

A **Route** tells CodeIgniter which **controller function** to run when a user visits a certain URL.

Think of it as a **map** that connects website links (like /resident/create) to functions inside your controller (like Resident::create()).

How to Add Routes

Open this file in your project:

```
app/Config/Routes.php
```

Scroll down to the part where you can define your own routes, and add this:

```php
$routes->get('/resident', 'Resident::index');
$routes->get('/resident/create', 'Resident::create');
$routes->post('/resident/store', 'Resident::store');
$routes->get('/resident/edit/(:num)', 'Resident::edit/$1');
$routes->post('/resident/update/(:num)', 'Resident::update/$1');
$routes->get('/resident/delete/(:num)', 'Resident::delete/$1');
```

What These Routes Do:

URL	Action
/resident	Show list of residents
/resident/create	Show form to add new resident
/resident/store	Save new resident (via POST)
/resident/edit/{id}	Show form to edit resident
/resident/update/{id}	Update resident (via POST)
/resident/delete/{id}	Delete resident

Your routes are now connected!

Step 8: Creating Views (Part of Activity 5)

note: Creating the UI (views) was already part of **Activity 5**. If you already designed the interface there, we would now connect it to make it functional.

What is a View?

A **View** in CodeIgniter is a file that contains **HTML code**. It's what the user **sees** on the browser.

- Views are used to display forms, tables, and other visual elements.
- They **receive data from the controller** and show it in a human-friendly way.

Where to Place the Views

Save your view files in this folder:

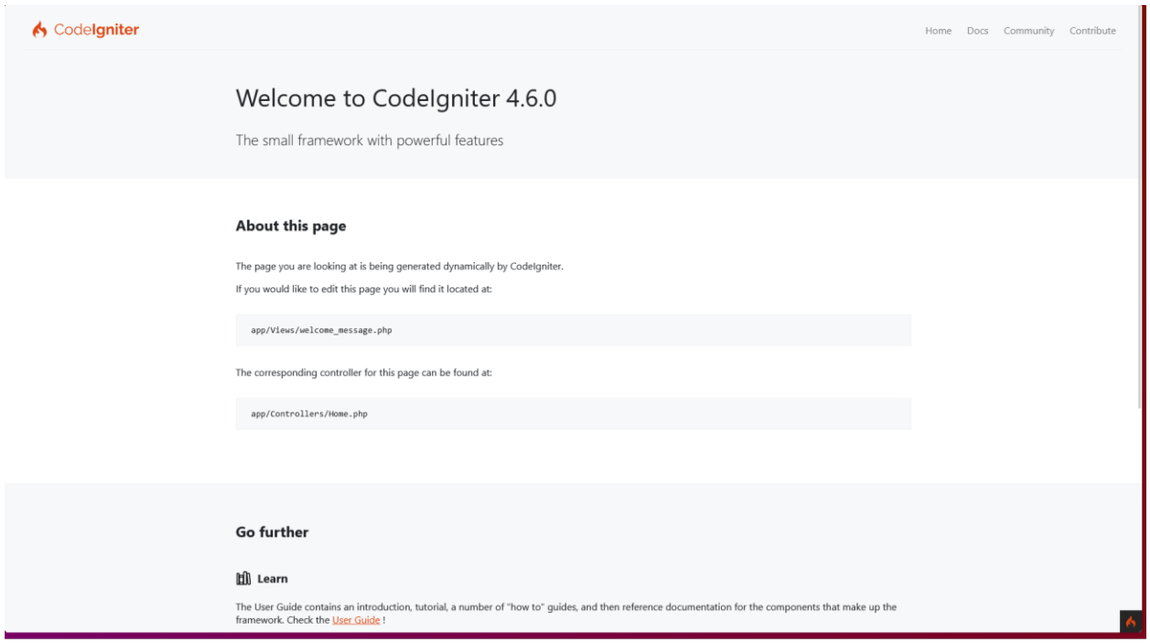
```
app/Views/residents/
```

Make sure you create the residents folder if it doesn't exist yet.

Save the file, restart the server (php spark serve), and visit:

```
C:\xampp\htdocs\BarangayIMS>php spark serve

CodeIgniter v4.6.0 Command Line Tool - Server Time: 2025-05-01 10:27:20 UTC+00:00

CodeIgniter development server started on http://localhost:8080
Press Control-C to stop.
[Thu May  1 18:27:21 2025] PHP 8.2.12 Development Server (http://localhost:8080) started
```

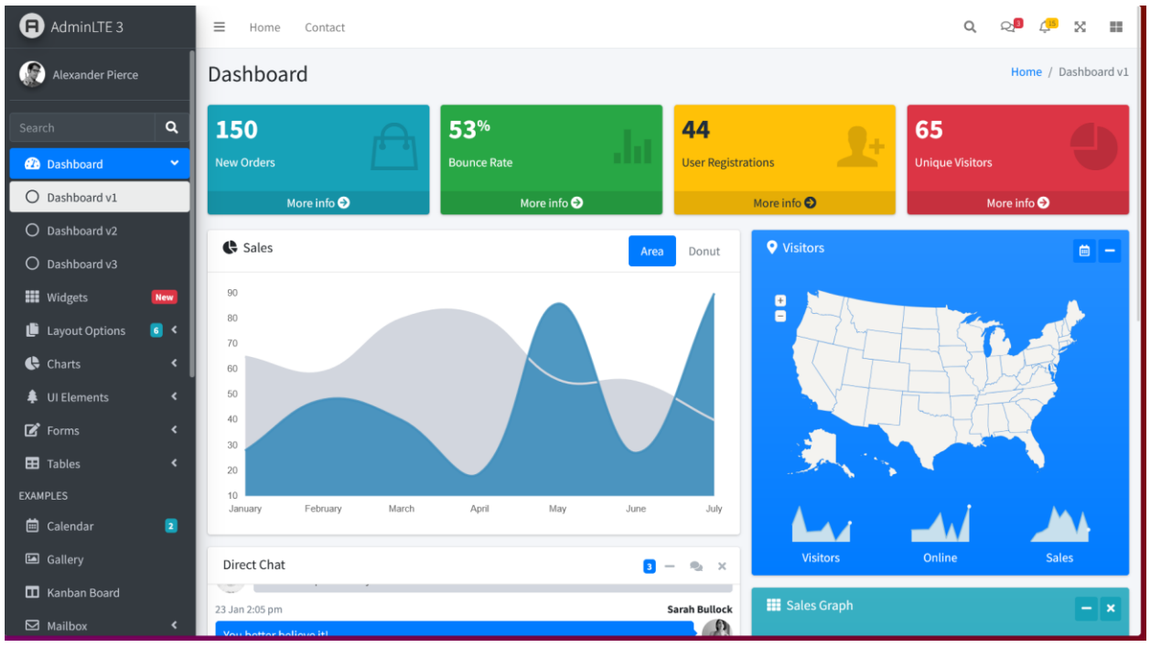
Open your browser and go to http://localhost:8080. You should see the CodeIgniter welcome page.



This is the UI from AdminLTE that I provided.

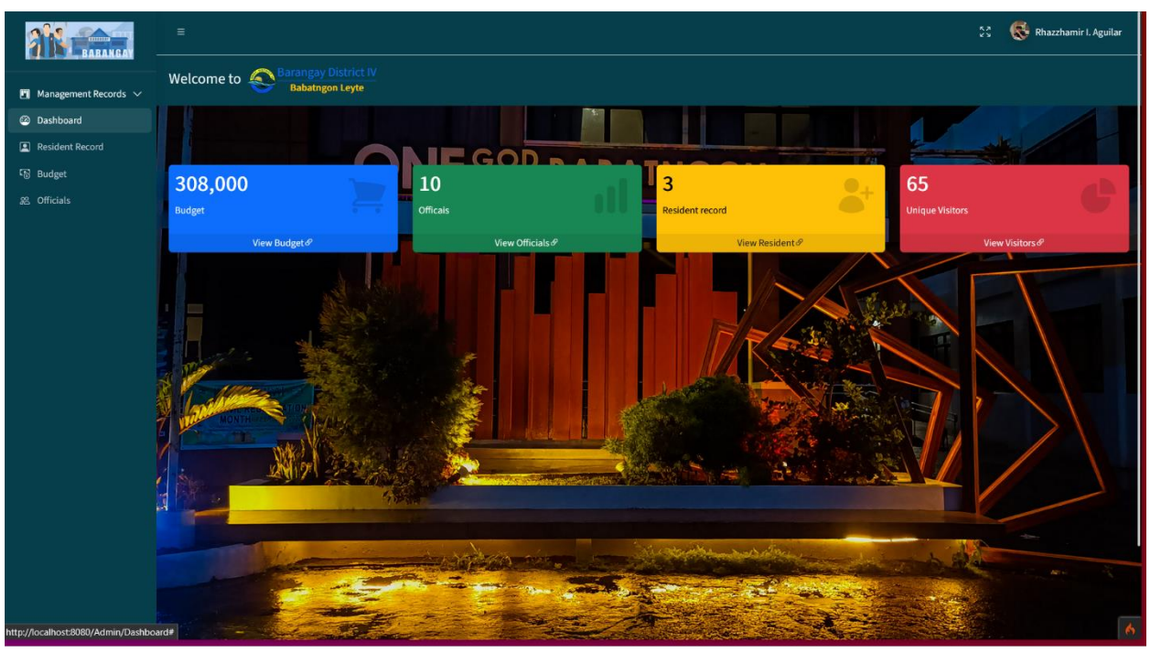
We are now using it as part of our project. Below is the **before and after** modification of the UI:

Before (Default AdminLTE UI)




```
<!-- Example AdminLTE card before modification -->
<div class="card">
  <div class="card-header">
    <h3 class="card-title">Default Title</h3>
  </div>
  <div class="card-body">
    <!-- Default content -->
  </div>
</div>
```

After (Modified to fit our Resident CRUD system)



```php
<div class="card">
  <div class="card-header">
    <h3 class="card-title">Resident List</h3>
    <div class="card-tools">
      <a href="/resident/create" class="btn btn-primary btn-sm">Add Resident</a>
    </div>
  </div>
  <div class="card-body">
    <table class="table table-bordered table-hover">
      <thead>
        <tr>
          <th>ID</th>
          <th>Full Name</th>
          <th>Age</th>
          <th>Gender</th>
          <th>Contact</th>
          <th>Action</th>
        </tr>
      </thead>
      <tbody>
        <?php foreach ($residents as $resident): ?>
        <tr>
          <td><?= $resident['id']; ?></td>
          <td><?= $resident['full_name']; ?></td>
          <td><?= $resident['age']; ?></td>
          <td><?= $resident['gender']; ?></td>
          <td><?= $resident['contact']; ?></td>
          <td>
            <a href="/resident/edit/<?= $resident['id']; ?>" class="btn btn-warning btn-sm">Edit</
            <a href="/resident/delete/<?= $resident['id']; ?>" class="btn btn-danger btn-sm" oncl
          </td>
        </tr>
        <?php endforeach; ?>
      </tbody>
    </table>
  </div>
</div>
```

This modified layout fits nicely with your AdminLTE design and makes your CRUD UI more modern and user-friendly.

After that, Let's proceed to the objective of the proposal with the interaction between the Admin and the User in your Barangay Certificate Request system, and build it based on your CRUD app in CodeIgniter 4.

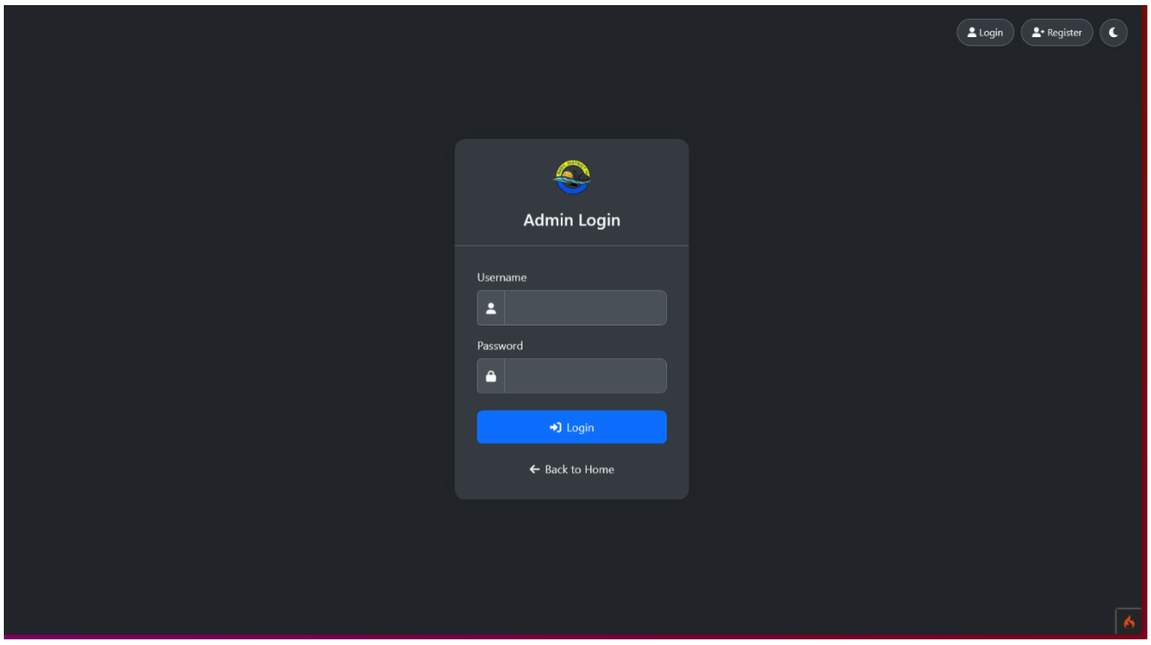
Title Proposal Objective – What You See Inside

1. ***Barangay Budget Management*** – To provide a structured system for recording, tracking, and managing the barangay's allocated budget, expenses, and financial reports.
2. ***Barangay Member and Officials Directory*** – To maintain an updated database of barangay officials, staff, and community leaders with their respective roles and responsibilities.
3. **Resident Records Management** – To create a secure and organized digital record-keeping system for all residents, including personal information, household data, and demographic details.

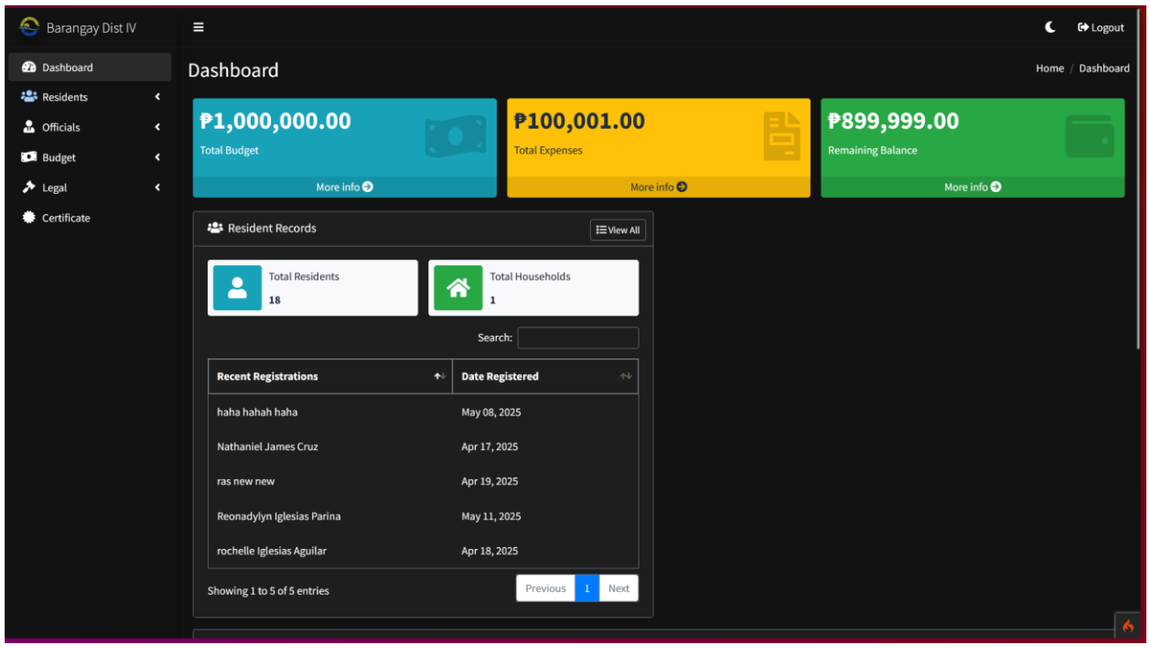
Before we proceed to the Admin dashboard we need to login to the admin form

Admin Log In

This login screen is only for the Barangay official or assigned Barangay members.
They need special credentials (admin access) to enter.

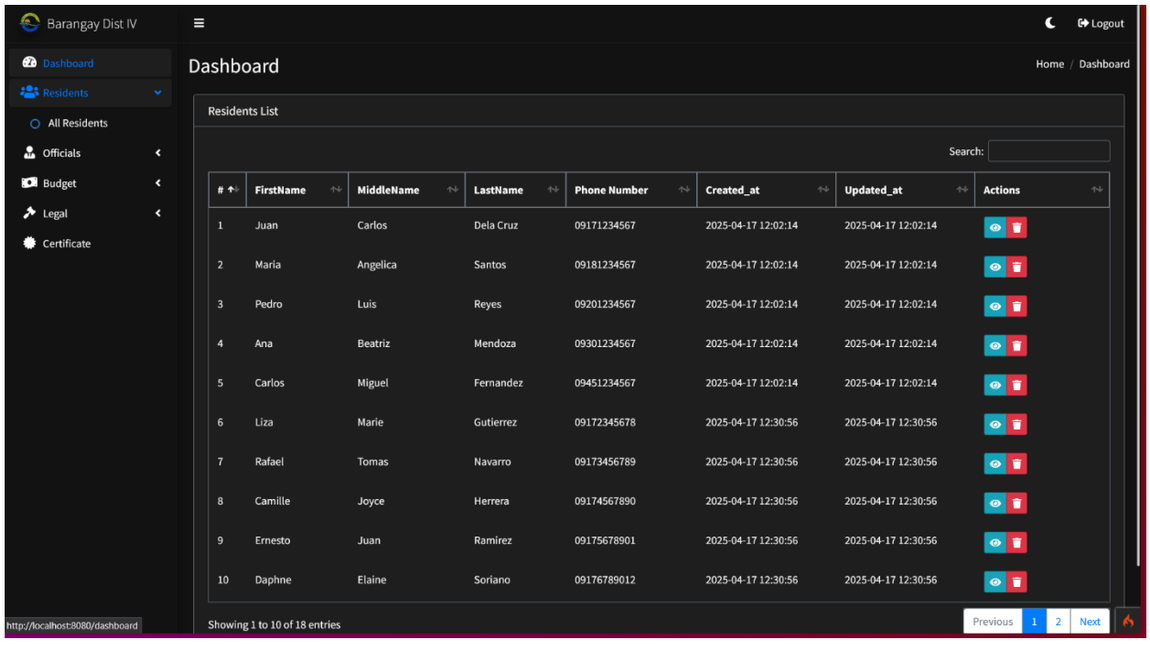


After logging in, the admin sees a control panel with many tools. They can manage and update the system.



Resident Management

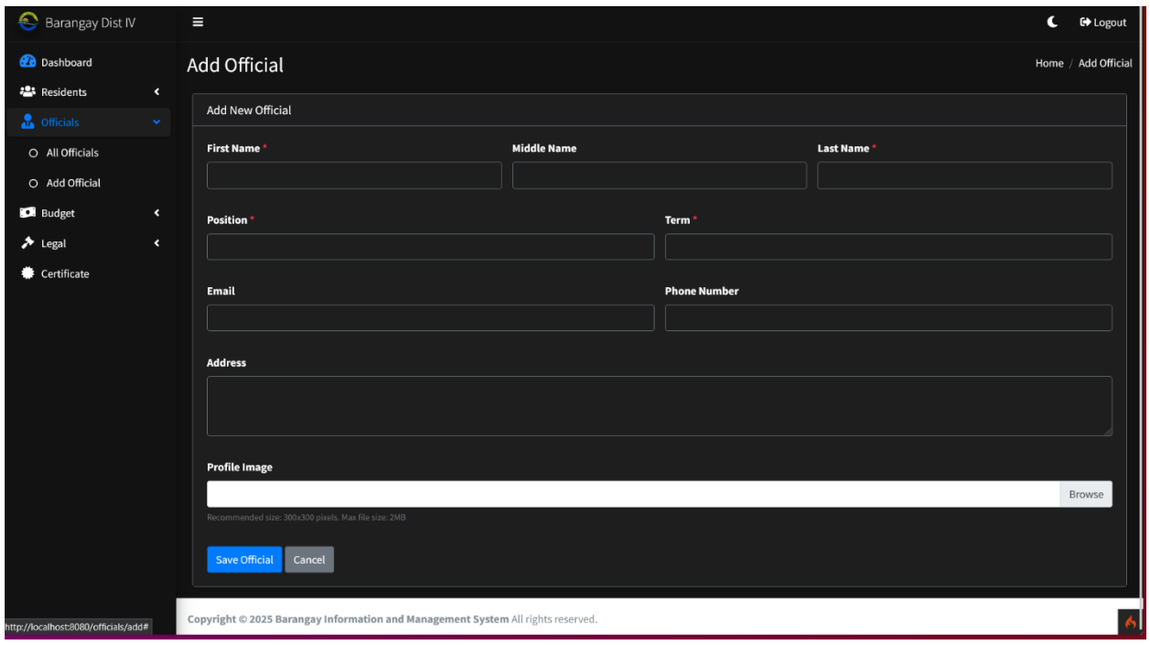
- *Admin can view resident information, remove inactive users.*



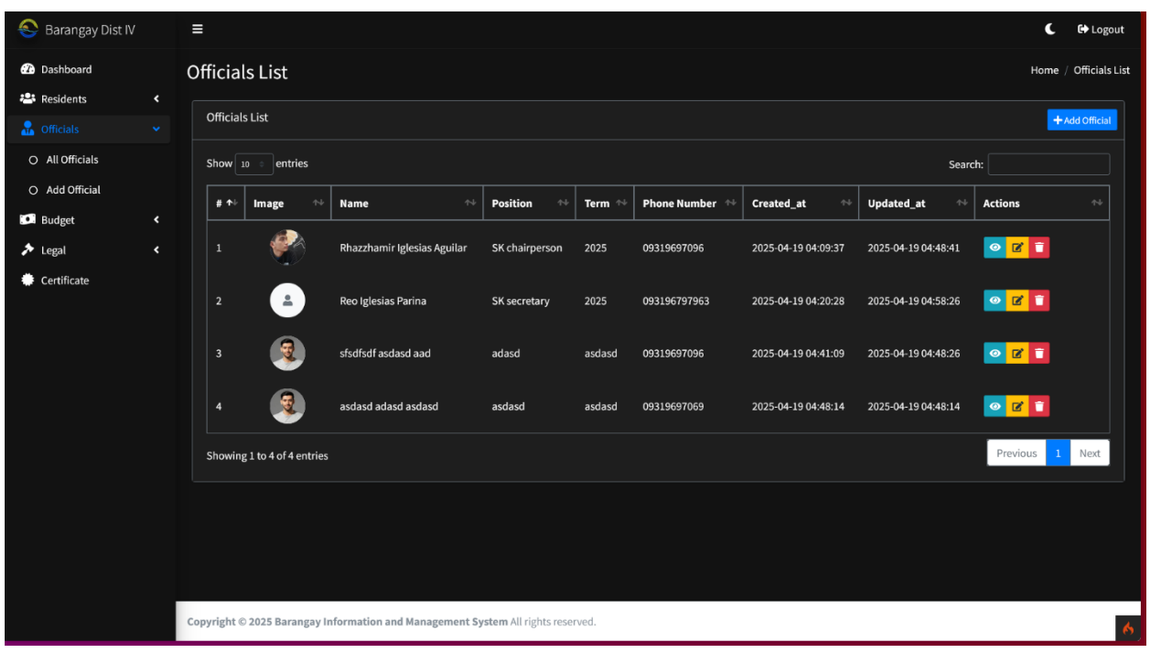
Official Management

- The admin can add a new Barangay official when the current official's term has ended.

Add official



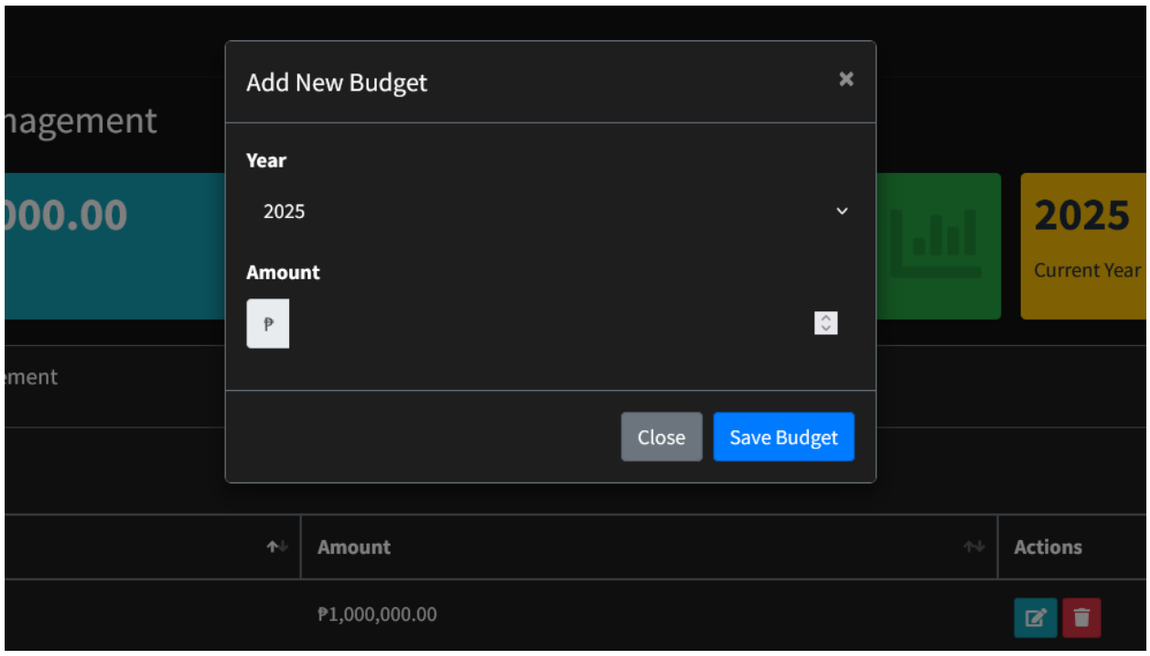
List of official



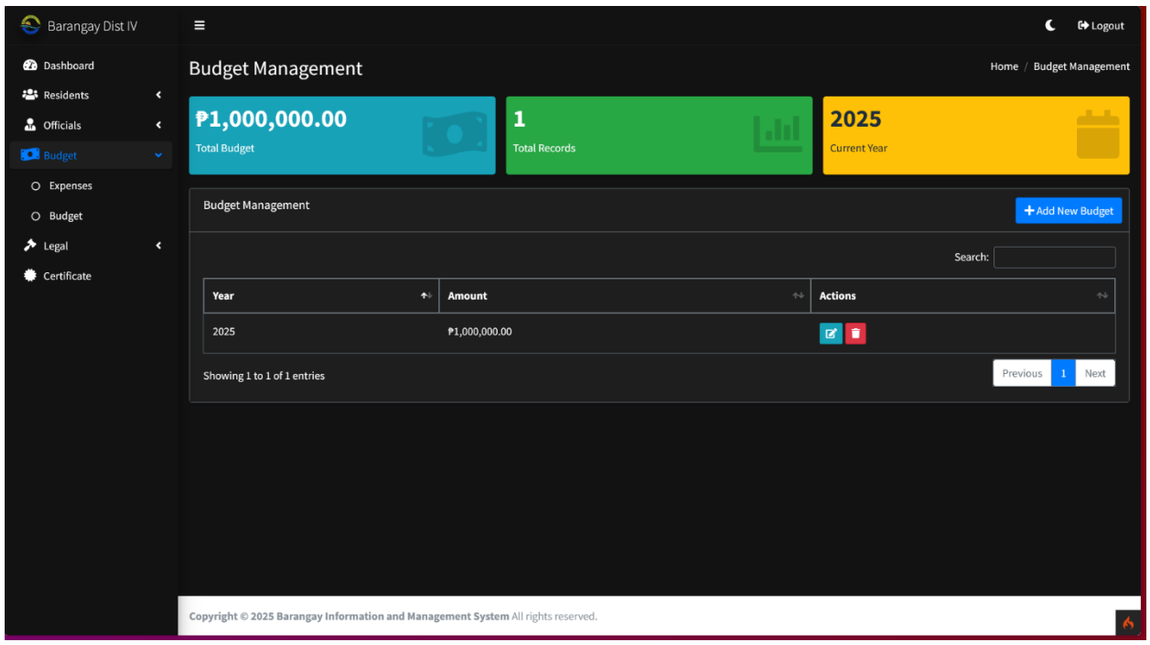
Budget Management

- Module for tracking and controlling barangay budget allocations.

Add budget

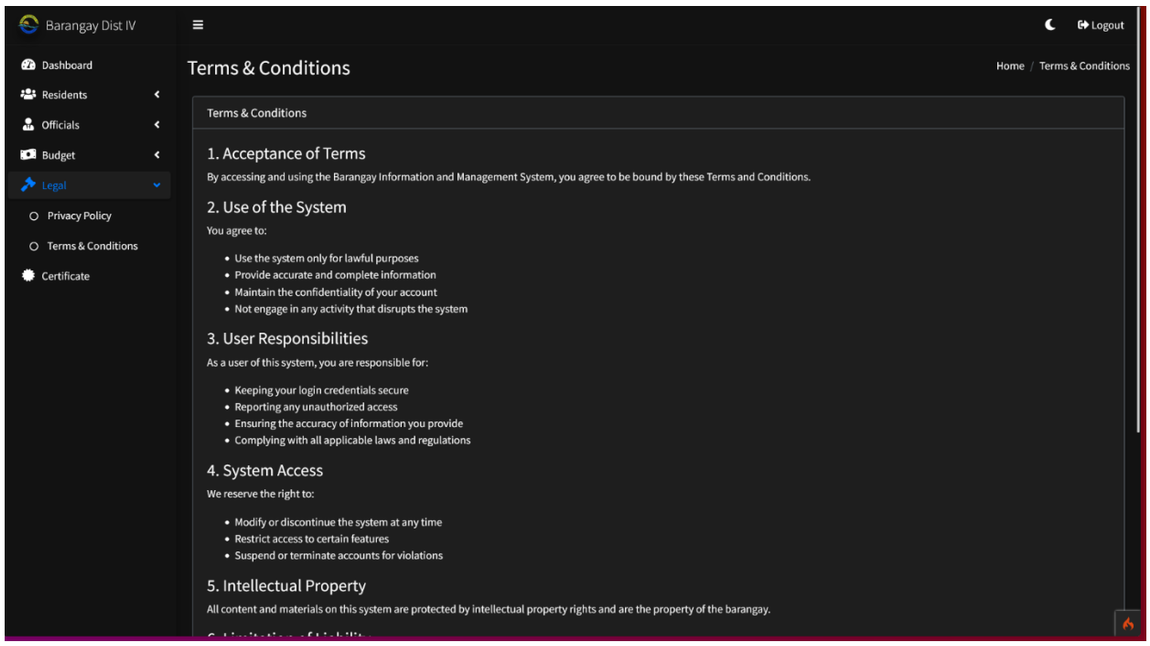


Record Budget



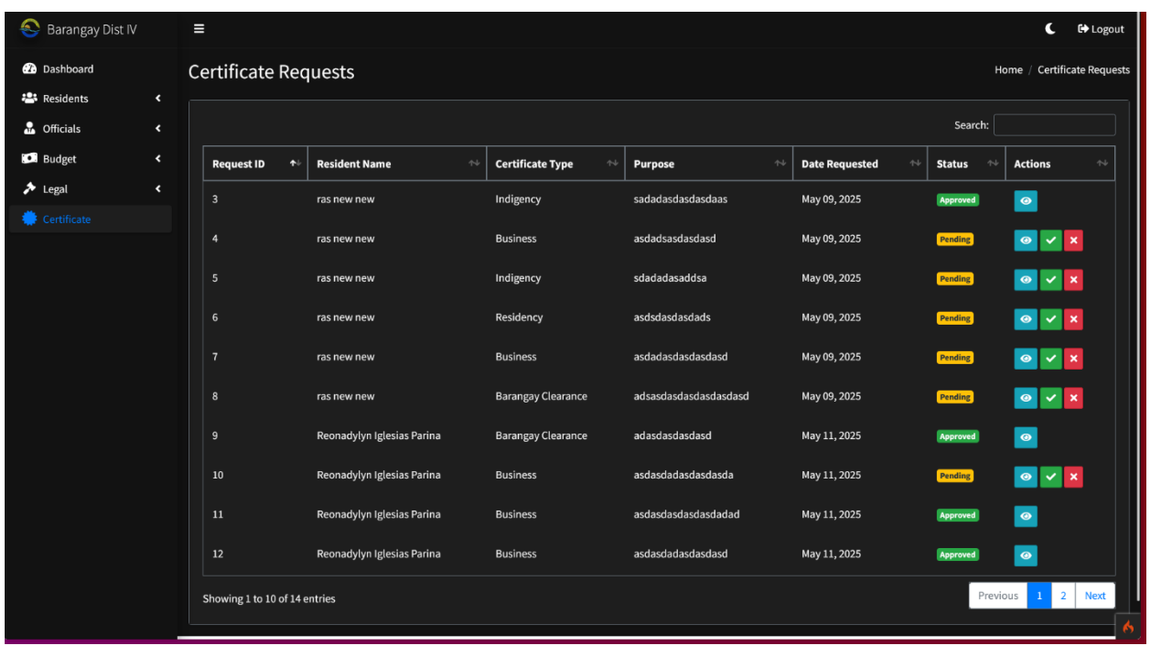
Legal

- Explains how your data is collected, used, protected, and your rights within the Barangay Information and Management System.

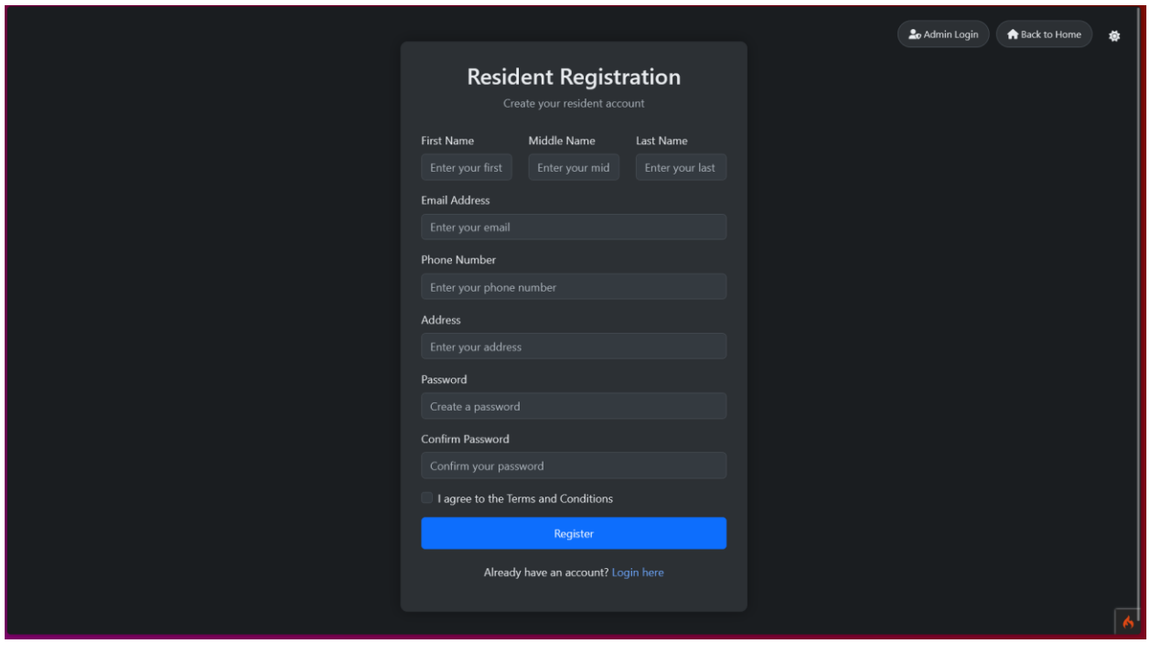


Certificate Management

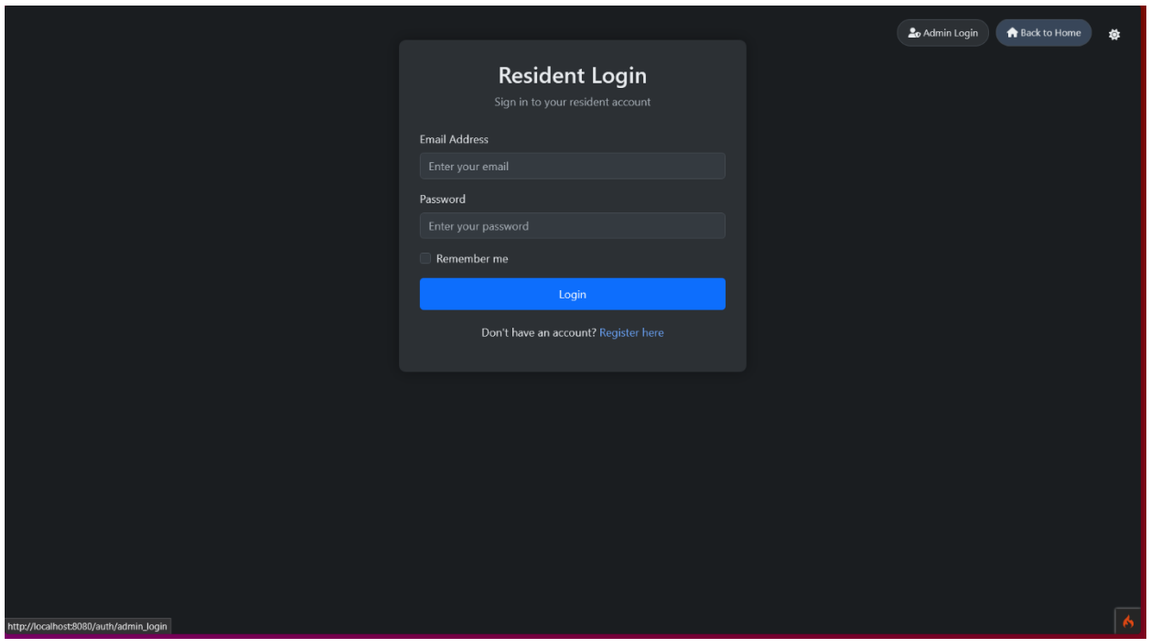
- Allows residents to request and official barangay can approved the request.



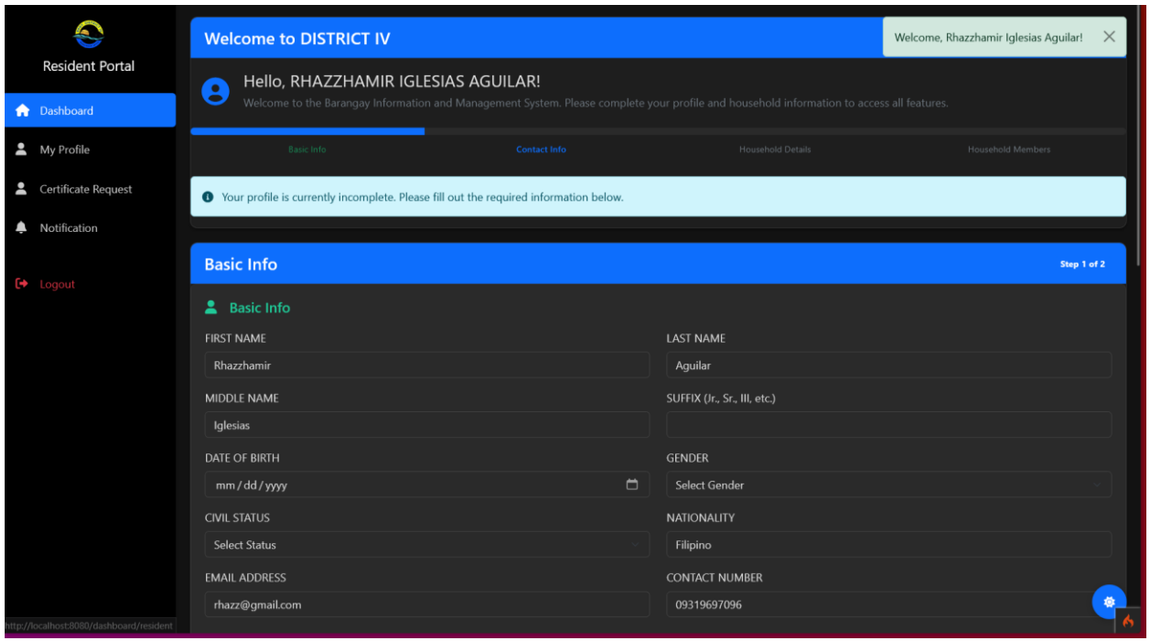
Before we proceed to the User dashboard we need to register to the registration



After register need to login

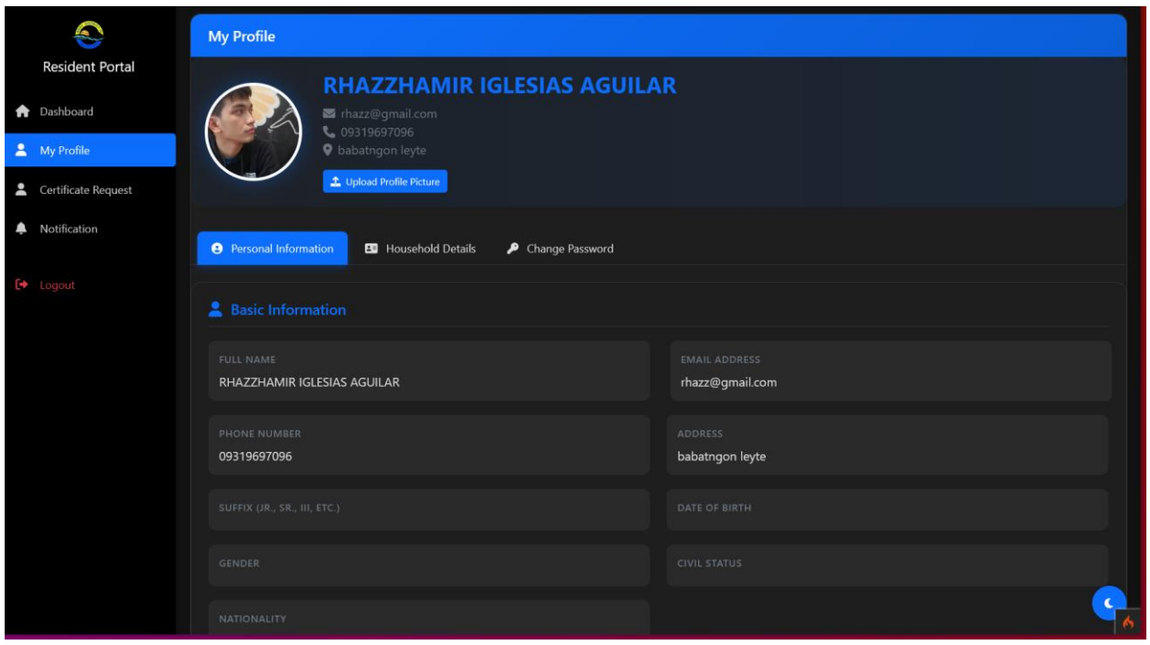


After logging in, the user sees a control panel with many tools. They can manage and update the information.



Profile Management

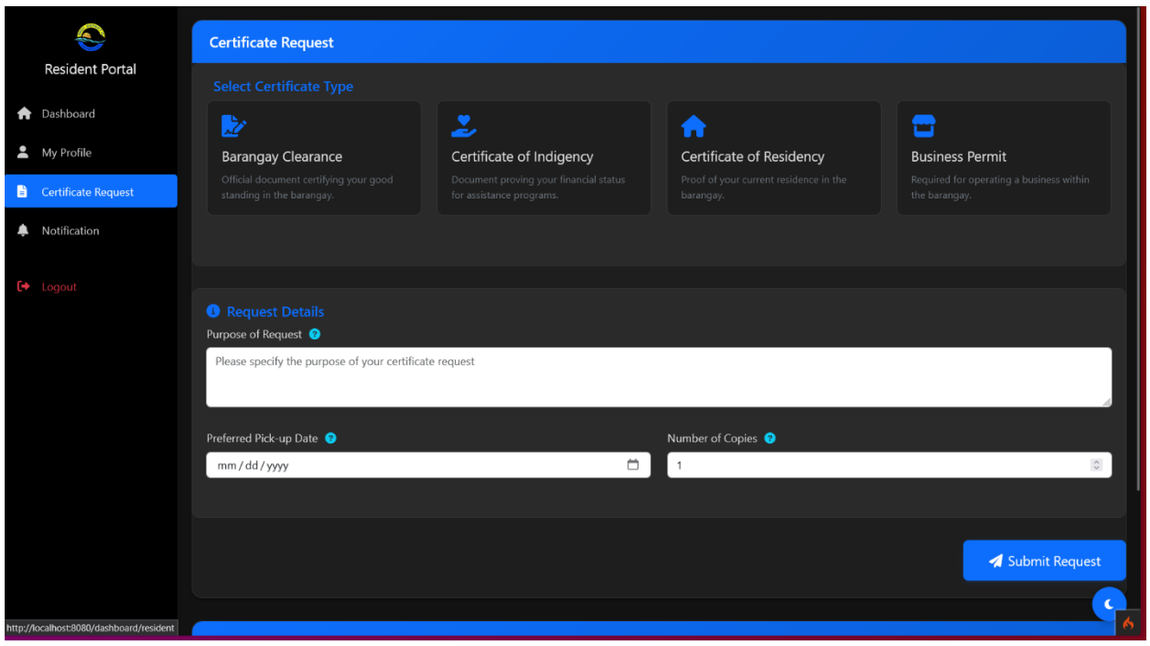
- *User can add or change their information and also the profile*



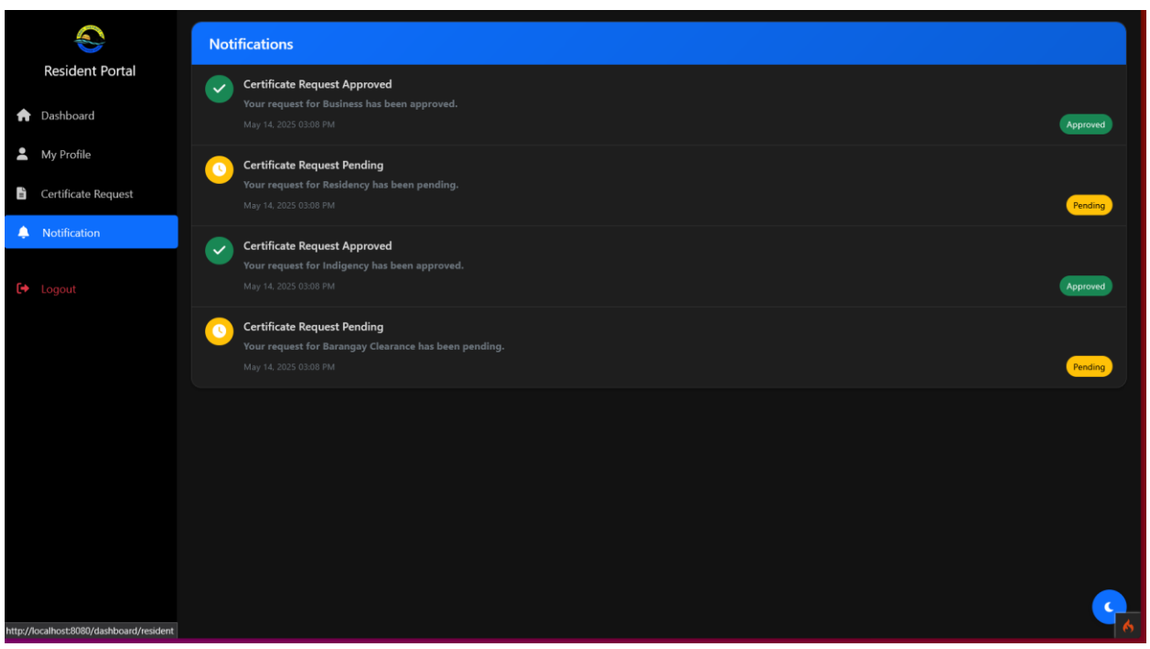
Certificate Request

- Resident can request a any types certificate and wait to approved from the admin

Certificate form



Notification when the certificate is approved



Conclusion

- Display a list of residents
- Allow adding new residents
- Allow editing resident details
- Allow deleting residents
- Allow adding new information
- Title Proposal Objective