



اوپنورسیتی ملیسيا قهغ السلطان عبد الله
UNIVERSITI MALAYSIA PAHANG
AL-SULTAN ABDULLAH

BCS3453

INTEGRATED APPLICATION DEVELOPMENT FRAMEWORK (PROJECT)

SESSION 2023/2024 SEMESTER I

| | |
|-----------------|-------------------------------------|
| LECTURER'S NAME | : TS. DR. ANIS FARIHAN MAT RAFFEI |
| NAME | : SYAZWANI NADHIRAH BINTI ZOLKEFILE |
| MATRIC NO | : CB21145 |
| PROJECT NAME | : TERATAI PINTAR WEB APPLICATION |

TABLE OF CONTENTS

| | |
|--|-----|
| TABLE OF CONTENTS | ii |
| LIST OF FIGURES | iii |
| CHAPTER 1 | 1 |
| 1.1 PROJECT BACKGROUND..... | 1 |
| 1.2 SYSTEM OVERVIEW..... | 3 |
| 1.3 SYSTEM PLAN | 4 |
| CHAPTER 2 | 5 |
| 2.1 DATABASE CONNECTION AND MIGRATION | 5 |
| 2.2 CONTROLLER ENTRY POINT AND USAGE | 7 |
| 2.3 MODEL MANIPULATION | 11 |
| 2.4 ROUTE OF WEB APPLICATION | 12 |
| 2.5 BOOTSTRAP/TAILWIND/VUE.JS INTEGRATION | 13 |
| 2.6 MIDDLEWARE AND CUSTOM MIDDLEWARE..... | 15 |
| 2.7 REGISTRATION AND LOGIN | 17 |
| 2.8 WEB-APPLICATION INTEGRATION | 19 |
| 2.9 ENTITY RELATIONSHIP DIAGRAM | 20 |
| CHAPTER 3 | 22 |
| 3.1 USE CASE DIAGRAM..... | 22 |
| 3.2 UNIFIED MODELING LANGUAGE (UML) DIAGRAM..... | 25 |
| 3.3 CONTROLLER FUNCTION | 28 |
| 3.4 OBJECT RELATIONAL MAPPING (ORM) | 42 |
| 3.5 VIEW ARRANGEMENT AND LAYOUT STRUCTURE | 46 |
| 3.6 DEPLOYMENT IN INDAH SERVER | 58 |

LIST OF FIGURES

| | |
|---|----|
| Figure 2. 1 env.file | 5 |
| Figure 2. 2 seeder file..... | 6 |
| Figure 2. 3 Laragon (Heidi SQL)..... | 6 |
| Figure 2. 4 Controller File | 7 |
| Figure 2. 5 Controller..... | 7 |
| Figure 2. 6 Application Controller..... | 8 |
| Figure 2. 7 ContactUsController..... | 8 |
| Figure 2. 8 EventController..... | 9 |
| Figure 2. 9 HomeController | 9 |
| Figure 2. 10 ProgramClassContoller..... | 10 |
| Figure 2. 11 Model File | 11 |
| Figure 2. 12 web.php | 12 |
| Figure 2. 13 Layout folder | 14 |
| Figure 2. 14 Route group | 15 |
| Figure 2. 15 kernel.php | 16 |
| Figure 2. 16 command 1 | 17 |
| Figure 2. 17 Command 2 | 18 |
| Figure 2. 18 Command 3 | 18 |
| Figure 2. 19 Command 3 | 19 |
| Figure 2. 20 Entity Relationship Diagram (ERD)..... | 20 |
| Figure 3. 1 Use Case Diagram | 22 |
| Figure 3. 2 UML Diagram | 25 |
| Figure 3. 3 ApplicationController..... | 29 |
| Figure 3. 4 ContactUsController..... | 31 |
| Figure 3. 5 EventController | 34 |
| Figure 3. 6 HomeController | 36 |
| Figure 3. 7 ProgramClassController | 38 |
| Figure 3. 8 ProgramController..... | 40 |
| Figure 3. 9 Application Model..... | 42 |
| Figure 3. 10 Blade file for edit application | 44 |
| Figure 3. 11 view page..... | 45 |
| Figure 3. 12 User Folder | 46 |
| Figure 3. 13 Admin Folder..... | 46 |
| Figure 3. 14 layouts folder | 47 |
| Figure 3. 15Auth Folder..... | 47 |
| Figure 3. 16 Landpage for guest | 47 |
| Figure 3. 17 user dashboard | 48 |
| Figure 3. 18 User program dashboard..... | 48 |
| Figure 3. 19 User program display..... | 49 |
| Figure 3. 20 User class dashboard | 49 |
| Figure 3. 21 user application form..... | 50 |
| Figure 3. 22 user application list..... | 50 |
| Figure 3. 23 User event dashboard..... | 50 |
| Figure 3. 24 User Contact Us..... | 51 |
| Figure 3. 25 Admin dashboard..... | 51 |
| Figure 3. 26 Admin program dashboard | 51 |
| Figure 3. 27 Admin program create | 52 |

| | |
|---|----|
| Figure 3. 28 Admin program display | 52 |
| Figure 3. 29 Admin program update..... | 53 |
| Figure 3. 30 Admin program delete | 53 |
| Figure 3. 31 Admin class dashboard..... | 53 |
| Figure 3. 32 Admin class create..... | 54 |
| Figure 3. 33 Admin class view | 54 |
| Figure 3. 34 Admin event create | 55 |
| Figure 3. 35 Admin application dashboard..... | 55 |
| Figure 3. 36 Admin application update..... | 56 |
| Figure 3. 37 Admin feedback dashboard | 56 |
| Figure 3. 38 Admin feedback view | 57 |
| Figure 3. 39 Admin user dashboard..... | 57 |
| Figure 3. 40 Indah server deployment | 58 |

CHAPTER 1

INTRODUCTION

1.1 PROJECT BACKGROUND

Teratai Pintar is a kindergarten system that focuses on tackling the current difficulties encountered by early childhood education institutions. One of the urgent concerns in the current ever-changing educational environment is the requirement for streamlined registration and enrolment procedures. Due to the inconvenience and time-consuming nature of old techniques, both parents and administrators are increasingly seeking a more efficient alternative that simplifies the enrolling process. "Teratai Pintar" endeavours to address this problem directly by offering a user-friendly platform that streamlines the registration process, ensuring it is easily accessible and convenient for parents while improving administrative effectiveness.

Furthermore, in the age of data-driven decision-making, proficient data administration and reporting have emerged as crucial components of educational institutions. The present challenge resides in effectively managing the intricate task of safely processing sensitive information about children and their families while adhering to privacy standards. "Teratai Pintar" recognizes this difficulty and strives to create a robust data management system, guaranteeing the privacy and protection of personal information. The solution will optimize administrative duties associated with data and offer extensive reporting features. "Teratai Pintar" provides parents and administrators with valuable information for making informed decisions and enhancing early childhood education. It is achieved by creating perceptive reports on enrolment trends. Essentially, "Teratai Pintar" is a comprehensive solution that not only tackles the present difficulties but also foresees and caters to the changing requirements of kindergarten systems in the digital era.

Objectives:

Enhanced User Experience: Simplify the kindergarten registration process, guaranteeing a user-friendly interface for parents and guardians. Streamline the process of submitting necessary documents and providing crucial information to improve the overall user experience.

Improved Accessibility: Allow parents and guardians to access the Teratai Pintar system conveniently. The system enhances accessibility and convenience by allowing users to register without needing to be physically present, which aligns with modern expectations.

Streamlining Operations: Enhance the administrative module in Teratai Pintar system to optimize the management of enrolment applications, manage ongoing event and streamline kindergarten registration processes. This module is designed to simplify processes for administrative officers in charge of the registration system.

1.2 SYSTEM OVERVIEW

Teratai Pintar is an integrated online registration system tailored to the kindergarten community. Teratai Pintar emphasizes two primary user roles: parents or guardians (users) and administrative officers (admin).

Modules and Functionalities:

1. Login and Registration Module

- a. Facilitates secure logins for all authorized users with role-based access.

2. Manage Program Module

- a. Admin can perform operations related to managing educational programs such as adding new programs and view list of programs.

3. Manage Classes

- a. Admin can handle the management of classes programs such as creating new classes and view class information.

4. Manage Application

- a. Admin can access and review applications submitted by users.
- b. Users can apply for enrollment.

5. Manage Event

- a. Admins organize and maintain information about various events.
- b. Users explore and view details about upcoming events.

6. Manage Feedback

- a. Admin can access and review feedback provided by users.
- b. Users can offer feedback on various aspects of the system.

7. Manage Users

- a. Admin handle user management within the system such as view listing of the user.

1.3 SYSTEM PLAN

Requirement Analysis:

- Identify the problem or need the software aims to address.
- Identify the specific roles of users and the functionalities of the system.
- Create the purpose and goals of the software.
- Defined objectives and expected outcomes.

Design:

- Create system architecture to view their interactions.
- Evaluate and determine the necessary technologies for the implementation of software (Visual Studio Code), backend (Laravel framework, Laragon database), frontend (HTML, CSS, JavaScript), and Bootstrap.

Implementation:

- Programming languages, frameworks, and tools to be used.
- Version control strategy such as GitHub
- Create a comprehensive database schema outlining tables, relationships, and entities.

Server Setup and Configuration:

- Set up the server environment for hosting the Teratai Pintar system on Integrated Development and Application Hosting (InDAH) server.
- Configure security measures, backups, and performance optimizations.

Deployment:

- Deploy the application on InDAH server, ensuring proper functionality and accessibility.

CHAPTER 2

PROJECT PLANNING

2.1 DATABASE CONNECTION AND MIGRATION

This section outlines the essential steps for configuring the database connection in Laravel. It explains the crucial significance of Laravel's migration system, which streamlines the task of handling alterations to the database schema, guaranteeing uniformity across different contexts.

```
10
11  DB_CONNECTION=mysql
12  DB_HOST=127.0.0.1
13  DB_PORT=3306
14  DB_DATABASE=teratai-pintar
15  DB_USERNAME=root
16  DB_PASSWORD=
17
```

Figure 2. 1 env.file

- DB_CONNECTION: Specifies the type of database.
- DB_HOST: Indicates the host where the database server is located.
- DB_PORT: Specifies the port for the database connection.
- DB_DATABASE: Represents the name of the database used by the Laravel application.
- DB_USERNAME: Specifies the username for accessing the database.
- DB_PASSWORD: Indicates the password associated with the specified database username.

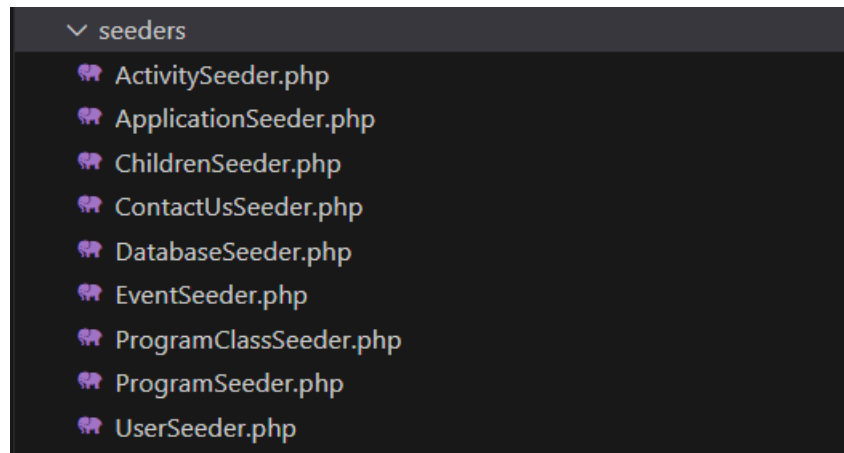


Figure 2. 2 seeder file

The 'database/seeders' directory contains important seeder files needed to fill the database with sample or starting data. These help with many parts of the app, like user data, programs, events, child information, and more, making the database setting in the Laravel project well-organized and valuable.

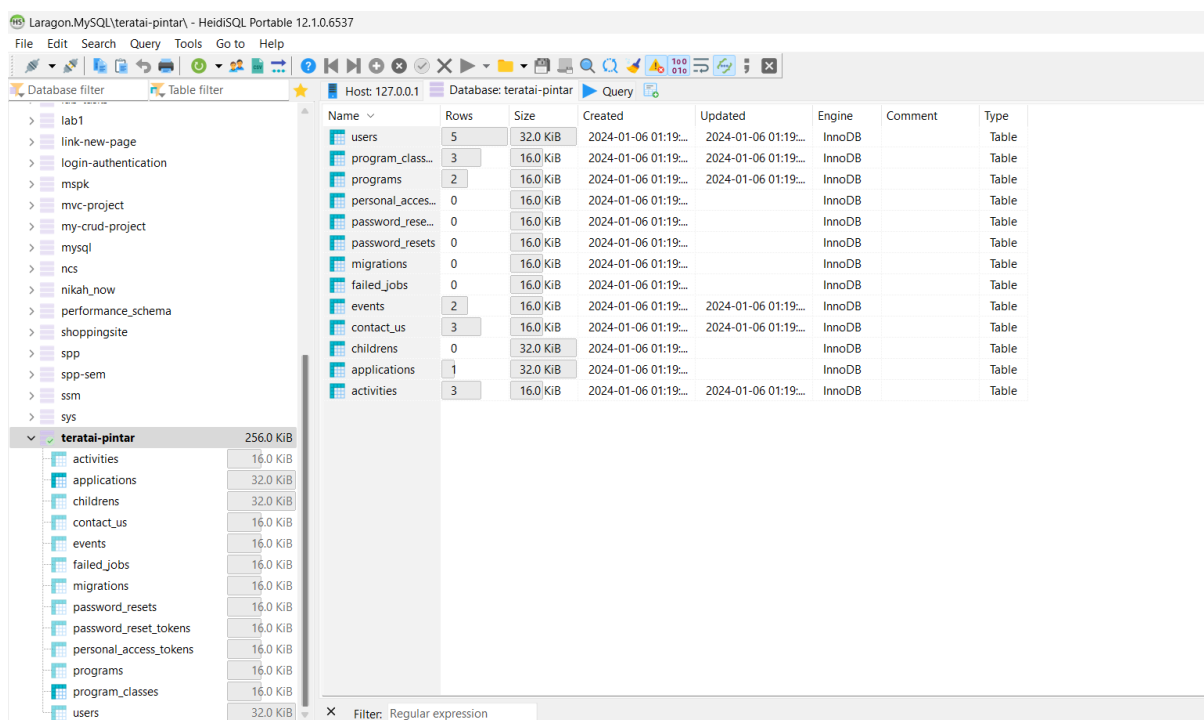


Figure 2. 3 Laragon (Heidi SQL)

Teratai Pintar system used Laragon database to store the data. The main section displays various tables within the 'teratai-pintar' databases, such as activities, applications and childrens.

2.2 CONTROLLER ENTRY POINT AND USAGE

Every Controller file has a unique role in Laravel Project. They are responsible for organising and managing various aspects of the applications logic and interaction.

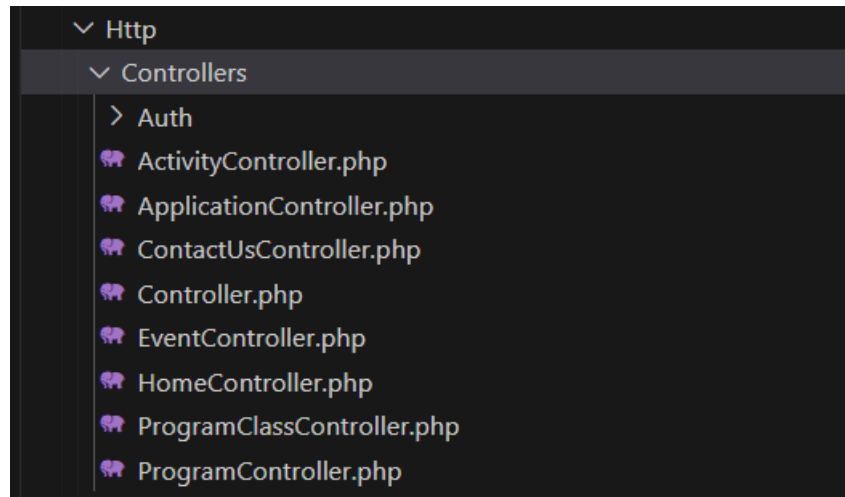


Figure 2. 4 Controller File

These controllers' files reside in the 'app\Http\Controllers' directory.

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
6  use Illuminate\Foundation\Validation\ValidatesRequests;
7  use Illuminate\Routing\Controller as BaseController;
8
9  You, 5 days ago | 1 author (You)
10 class Controller extends BaseController
11 {
12     use AuthorizesRequests, ValidatesRequests;
13 }
```

Figure 2. 5 Controller

Start with the Controller.php file, which is the primary controller for the Laravel project. Other controllers use it as a reference. This base controller is like a model; it provides essential features that can be used in other controllers in the Laravel application. It makes sure that best practices are followed when validating and authorizing requests.

```
class ApplicationController extends Controller
{
  /**
   * Display a listing of the resource.
   */
  public function index()
  {
    $user = User::where('id', auth()->user()->id)->first();
    $applications = Application::whereHas('user', function ($query) {
      $query->where('user_id', auth()->user()->id);
    }->paginate(10);

    return view('user.application.index', compact('user', 'applications'));
  }
}
```

Figure 2. 6 Application Controller

This ApplicationController specifically managing user applications for program enrolment. This controller efficiently manages the CRUD (Create, Read, Update, Delete) operations for user applications within a kindergarten system. It ensures proper relationships between users, children, programs, classes, and activities while providing distinct views for both users and administrators.

```
9 class ContactUsController extends Controller
10 {
11   /**
12    * Display a listing of the resource.
13    */
14   public function index()
15   {
16     $user = User::where('id', auth()->user()->id)->first();
17     $contactUs = ContactUs::all();
18     return view('admin.contactUs.index', compact('contactUs', 'user'));
19   }
20 }
21
```

Figure 2. 7 ContactUsController

This ContactUsController efficiently manages the interaction between users and the system's contact form. It captures user inquiries, associates them with the authenticated user, and allows administrators to view these messages.

```
8 class EventController extends Controller
9 {
10     /**
11      * Display a listing of the resource.
12      */
13     public function index()
14     {
15         $user = User::where(auth()->user()->role == '1');
16         $events = Event::All();
17         return view('admin.event.index', compact('events'));
18     }
19 }
```

Figure 2. 8 EventContoller

This EventController efficiently manages event-related operations, including displaying events for users and administering events for administrators.

```
12 class HomeController extends Controller
13 {
14     /**
15      * Create a new controller instance.
16      *
17      * @return void
18      */
19     public function __construct()
20     {
21         $this->middleware('auth');
22     }
23
24 }
```

Figure 2. 9 HomeController

This HomeController efficiently manages various aspects of the user interface, providing distinct views for both regular users and administrators. It also offers functionalities to view programs, classes, and user details, contributing to a comprehensive dashboard experience.

```
9 class ProgramClassController extends Controller
10 {
11     /**
12      * Display a listing of the resource.
13      */
14     public function index()
15     {
16         $user = User::where(auth()->user()->role == '1');
17         // $programs = Program::All();
18         $classes = ProgramClass::All();
19         return view('admin.class.index',compact('classes'));
20     }
21 }
```

Figure 2. 10 ProgramClassContoller

This ProgramClassController efficiently manages operations related to program classes, including displaying, creating, showing details, and deleting classes. The index() function provides an admin view with program class management functionalities

```
8 class ProgramController extends Controller
9 {
10     /**
11      * Display a listing of the resource.
12      */
13     public function index()
14     {
15         $user = User::where(auth()->user()->role == '1');
16         $programs = Program::All();
17         return view('admin.program.index',compact('programs'));
18     }
19 }
```

this ProgramController efficiently manages program-related operations, including displaying, creating, showing details, updating status, and deleting programs. The index() function provides an admin view with program management functionalities

2.3 MODEL MANIPULATION

Laravel models use the ORM paradigm for easy interaction with database tables using object-oriented methods.

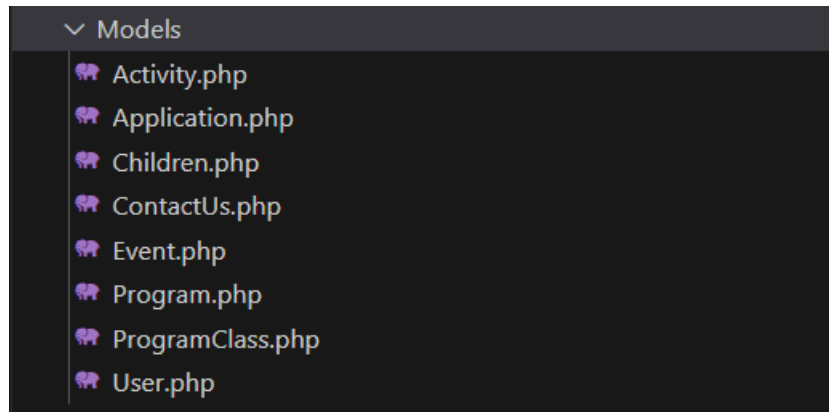


Figure 2. 11 Model File

By encapsulating the logic and operations associated with the corresponding database tables, each of these models provides a Laravel application with an object-oriented interface through which to interact with the underlying data.

2.4 ROUTE OF WEB APPLICATION

The routing system in Laravel is flexible because it supports named routes, route parameters, wildcards, middleware, and route groups.

```
//General User Routes
Route::middleware(['auth', 'user-role:user'])->group(function() {
    Route::get('/home', [HomeController::class, 'index'])->name('user.dashboard');
    Route::get('/user/program', [HomeController::class, 'program'])->name('user.program');
    Route::get('/user/program/show/{id}', [HomeController::class, 'showProgram'])->name('user.showProgram');
    Route::get('/user/class', [HomeController::class, 'class'])->name('user.class');
    Route::get('/user/class/show/{id}', [HomeController::class, 'showClass'])->name('user.showClass');
    Route::get('/user/event', [EventController::class, 'display'])->name('event.display');
    Route::resource('contactUs', ContactUsController::class);
    Route::resource('application', ApplicationController::class);
    Route::put('application/delete/{id}', [ApplicationController::class, 'destroy'])->name('application.destroy');
});
```

Figure 2. 12 web.php

This structure shows:

- Using identity and verification middleware to protect routes
- Adding custom middleware.

2.5 BOOTSTRAP/TAILWIND/VUE.JS INTEGRATION

Front-end integration within a Laravel project entails integrating robust front-end frameworks and tools like Bootstrap, Tailwind CSS, and Vue.js. These integrations aim to improve the user experience, simplify development, and enable interactive components in the program.

Bootstrap, a prevalent CSS framework, provides an extensive pre-styled element and a grid structure that adjusts to different screen sizes. By incorporating Bootstrap into Laravel, users can effortlessly generate uniform and aesthetically pleasing user interface components.

Tailwind CSS, renowned for its utility-first methodology, offers a versatile and adaptable means of styling interfaces. The integration of Laravel enables developers to create user interface components by directly applying utility classes within HTML templates.

Vue.js is a JavaScript framework that allows developers to create dynamic and responsive components. Integrating Vue.js within Laravel provides dynamic interfaces and smooth communication between the front and back end.

```

<head>
  <meta charset="utf-8">
  <meta content="width=device-width, initial-scale=1.0" name="viewport">
  <meta name="csrf-token" content="{{ csrf_token() }}">

  <title>Teratai Pintar</title>

  <!-- Favicons -->
  <link href="{{ asset('assets/img/logo.png') }}" rel="icon">
  <link href="{{ asset('assets/img/logo.png') }}" rel="logo">

  <!-- Google Fonts -->
  <link href="https://fonts.gstatic.com" rel="preconnect">
  <link
    href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,600,600i,700,700i|Nunito:300,300i,400,400i"
    rel="stylesheet">
  You, 4 days ago • make controller and migration file
  <!-- Vendor CSS Files -->

  <link href="{{ asset('assets/vendor/bootstrap/css/bootstrap.min.css') }}" rel="stylesheet">
  <link href="{{ asset('assets/vendor/bootstrap-icons/bootstrap-icons.css') }}" rel="stylesheet">
  <link href="{{ asset('assets/vendor/boxicons/css/boxicons.min.css') }}" rel="stylesheet">
  <link href="{{ asset('assets/vendor/quill/quill.snow.css') }}" rel="stylesheet">
  <link href="{{ asset('assets/vendor/quill/quill.bubble.css') }}" rel="stylesheet">
  <link href="{{ asset('assets/vendor/remixicon/remixicon.css') }}" rel="stylesheet">
  <link href="{{ asset('assets/vendor/simple-datatables/style.css') }}" rel="stylesheet">

```

Figure 2. 13 Layout folder

The baseUser.blade.php and adminBase.blade.php file is a template for the organization and visual design of web pages across the application. Additional Blade views utilize and inherit from this layout to provide uniformity throughout the website and prevent code repetition.

Various **<script>** elements that include external JavaScript files from vendor libraries. These files provide functionality for charts, Bootstrap, Chart.js, ECharts, Quill, Simple DataTables, and more. Thus, **<link>** elements that include external CSS files from vendor libraries. These files provide styling for elements on the page. Notable ones include Bootstrap, Bootstrap Icons, Boxicons, Quill, Remixicon, and Simple DataTables.

2.6 MIDDLEWARE AND CUSTOM MIDDLEWARE

Custom middleware is 'UserRole' where the middleware is responsible for checking the roles of an authenticated user before allowing access to specific routes. The 'userRole' middleware is designed to handle scenarios where different user roles (admin and user) would have different access permissions.

```

32 //General User Routes
33 Route::middleware(['auth', 'user-role:user'])->group(function() {
34     Route::get('/home', [HomeController::class, 'index'])->name('user.dashboard');
35     Route::get('/user/program', [HomeController::class, 'program'])->name('user.program');
36     Route::get('/user/program/show/{id}', [HomeController::class, 'showProgram'])->name('user.showProgram');
37     Route::get('/user/class', [HomeController::class, 'class'])->name('user.class');
38     Route::get('/user/class/show/{id}', [HomeController::class, 'showClass'])->name('user.showClass');
39     Route::get('/user/event', [EventController::class, 'display'])->name('event.display');
40     Route::resource('contactUs', ContactUsController::class);
41     Route::resource('application', ApplicationController::class);
42     Route::put('application/delete/{id}', [ApplicationController::class, 'destroy'])->name('application.destroy');
43
44 });
45
46 // Admin Routes
47 Route::middleware(['auth', 'user-role:admin'])->group(function() {
48     Route::get('/admin/home', [HomeController::class, 'indexAdmin'])->name('admin.dashboard');
49     Route::resource('program', ProgramController::class);
50     Route::put('program/delete/{id}', [ProgramController::class, 'destroy'])->name('program.destroy');
51     Route::get('/admin/contactUs', [ContactUsController::class, 'index'])->name('contactUs.index');
52     Route::resource('event', EventController::class);

```

Figure 2. 14 Route group

In these route groups, the user-role middleware is applied with the roles "user" and "admin" respectively. The middleware checks if the authenticated user has the expected role to access the routes within the group. If the check fails, it returns a JSON response indicating a lack of permission.

```
protected $middlewareAliases = [  
    'auth' => \App\Http\Middleware\Authenticate::class,  
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,  
    'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,  
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,  
    'can' => \Illuminate\Auth\Middleware\Authorize::class,  
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,  
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,  
    'precognitive' => \Illuminate\Foundation\Http\Middleware\HandlePrecognitiveRequests::class,  
    'signed' => \App\Http\Middleware\ValidateSignature::class,  
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,  
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,  
    'user-role' => \App\Http\Middleware\UserRole::class,  
];
```

Figure 2. 15 kernel.php

This registration makes the 'UserRole' middleware available for use in the application. In addition to these custom middleware classes, there are standard Laravel middleware such as the auth middleware, which is used to ensure that only authenticated users can access certain routes.

2.7 REGISTRATION AND LOGIN

In the Teratai Pintar website, the integration of the **composer require laravel/ui** command plays a crucial role in expediting the development of user registration and login functionalities. This Laravel package, when combined with the **--auth** option, automates the creation of essential components needed for a robust authentication system on the Teratai Pintar website.

```
C:\laragon\www\teratai-pintar
λ composer require laravel/ui
./composer.json has been updated
Running composer update laravel/ui
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking laravel/ui (v4.3.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Downloading laravel/ui (v4.3.0)
  - Installing laravel/ui (v4.3.0): Extracting archive
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

 INFO  Discovering packages.

laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
laravel/ui ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
spatie/laravel-ignition ..... DONE
```

Figure 2. 16 command 1

```
C:\laragon\www\teratai-pintar
λ php artisan ui vue --auth

INFO Authentication scaffolding generated successfully.
INFO Vue scaffolding installed successfully.
WARN Please run [npm install && npm run dev] to compile your fresh scaffolding.

C:\laragon\www\teratai-pintar
λ npm install

added 55 packages, and audited 56 packages in 2m

9 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Figure 2. 17 Command 2

After executing **composer require laravel/ui**, we need to execute **php artisan ui bootstrap --auth** to generate the required scaffolding. This includes controllers, views, and routes necessary for user registration and login. The chosen Bootstrap frontend preset ensures a visually appealing and responsive design for a seamless user experience. **npm install** is executed to install the Node.js dependencies specified in the project's **package.json** file. This command is vital for installing the required frontend packages and dependencies. These can include Bootstrap, jQuery, or other tools necessary for creating a responsive and visually appealing user interface.

```
C:\laragon\www\teratai-pintar
λ npm run build

> build
> vite build

vite v5.0.10 building for production...
✓ 114 modules transformed.
public/build/manifest.json 0.26 kB | gzip: 0.14 kB
public/build/assets/app-lhA9k1qk.css 225.67 kB | gzip: 30.76 kB
public/build/assets/app-4BERAh6J.js 265.83 kB | gzip: 93.43 kB
✓ built in 6.61s

C:\laragon\www\teratai-pintar
λ php artisan serve

INFO Server running on [http://127.0.0.1:8000].

Press Ctrl+C to stop the server
```

Figure 2. 18 Command 3

After installing the frontend dependencies, **npm run build** ensures that the frontend assets are ready for deployment, enhancing website performance by minimizing file sizes and improving loading times. This step is crucial for delivering an efficient and responsive user interface to visitors.

```
>
PS C:\laragon\www\teratai-pintar> php artisan migrate

INFO Preparing database.

Creating migration table ..... 60ms DONE

INFO Running migrations.

2014_10_12_000000_create_users_table ..... 50ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 13ms DONE
2014_10_12_100000_create_password_resets_table ..... 26ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 26ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 47ms DONE

PS C:\laragon\www\teratai-pintar> |
```

Figure 2. 19 Command 3

Lastly, the **php artisan migrate** command, executed after the **composer require laravel/ui** command, plays a crucial role in setting up the database schema. This command runs any pending migrations, creating the necessary tables to store user information such as user table and other relevant data.

2.8 WEB-APPLICATION INTEGRATION

Web application integration seamlessly combines different web applications or services, sharing data and functionalities. In the context of Teratai Pintar, a web-based system, integration may involve connecting various application components to enhance overall functionality, user experience, and efficiency. Here's an explanation web application integration that are implemented in Teratai Pintar:

User Authentication and Authorization:

- Integration with a user authentication system ensures secure access to the application.

Event and Calendar Integration:

- It involves scheduling events integrated with calendar services such as Google Calendar. Users can sync their Teratai Pintar schedules with their calendar applications.

2.9 ENTITY RELATIONSHIP DIAGRAM

An Entity-Relationship Diagram (ERD) is a visual representation of the data model that represents the relationships between entities in a database.

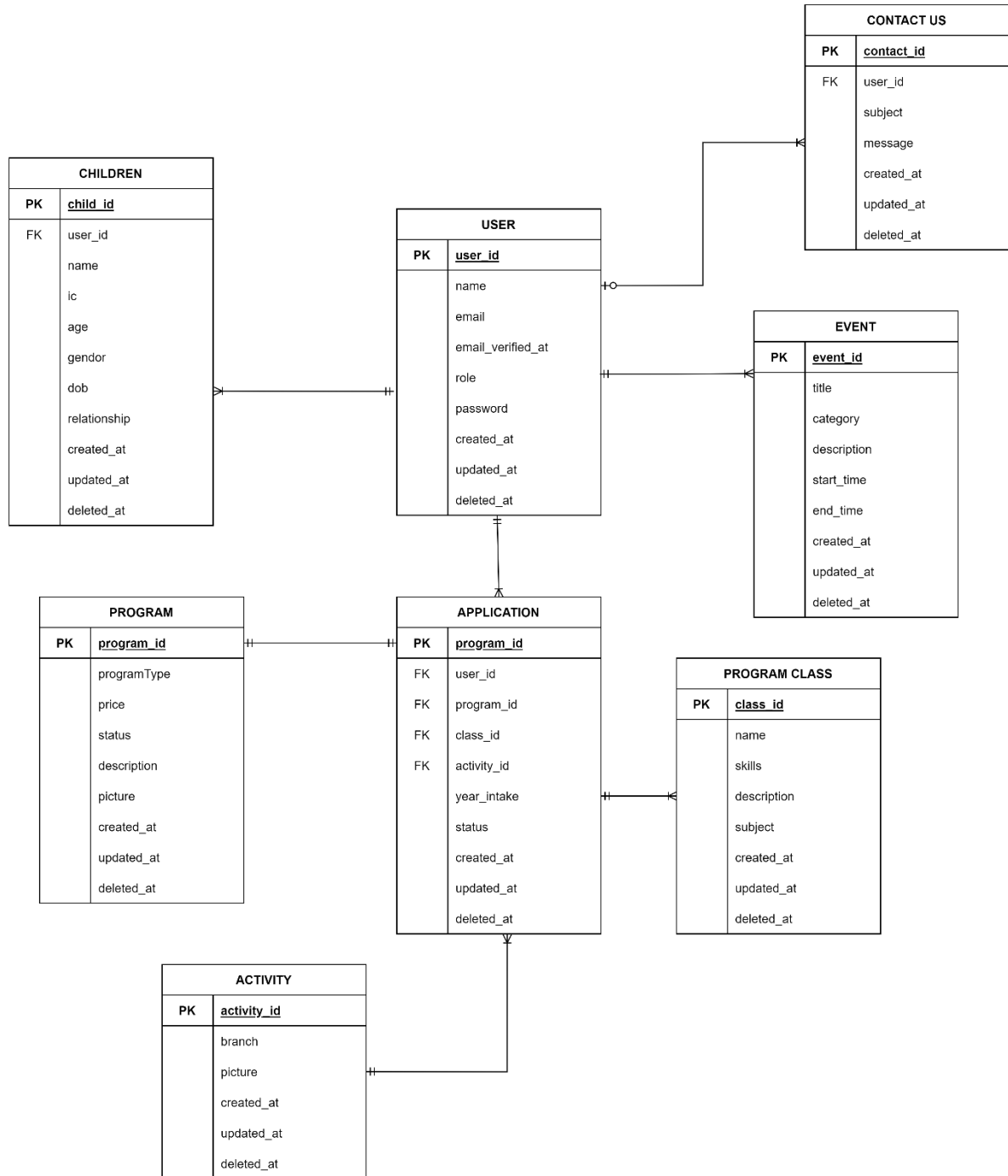


Figure 2. 20 Entity Relationship Diagram (ERD)

Relationships:

1. **User - Application (One-to-Many):**

- A user can make multiple applications, but each application belongs to only one user.

2. **Program - Application (One-to-Many):**

- A program can have multiple applications, but each application applies to only one program.

3. **ProgramClass - Application (One-to-Many):**

- A program class can have multiple applications, but each application is associated with only one program class.

4. **Activity - Application (One-to-Many):**

- An activity can be associated with multiple applications, but each application is linked to only one activity.

5. **User - Children (One-to-One):**

- Each user may have one or more children, but each child belongs to only one user.

6. **ContactUs - User (Many-to-One):**

- Multiple feedback entries can be associated with one user, indicating that a user might provide feedback multiple times.

7. **Event - User (Many-to-Many):**

- An event can be attended by multiple users, and a user can attend multiple events. This relationship is represented by a junction table to handle the many-to-many relationship.

8. **Program - ProgramClass (One-to-Many):**

- A program can have multiple classes, but each class is associated with only one program.

CHAPTER 3

IMPLEMENTATION

3.1 USE CASE DIAGRAM

A Use Case Diagram provides a visual representation of the functional requirements of a system from the end user's perspective. It illustrates the system's intended functionality by showing how users will interact with the system and its various use cases.

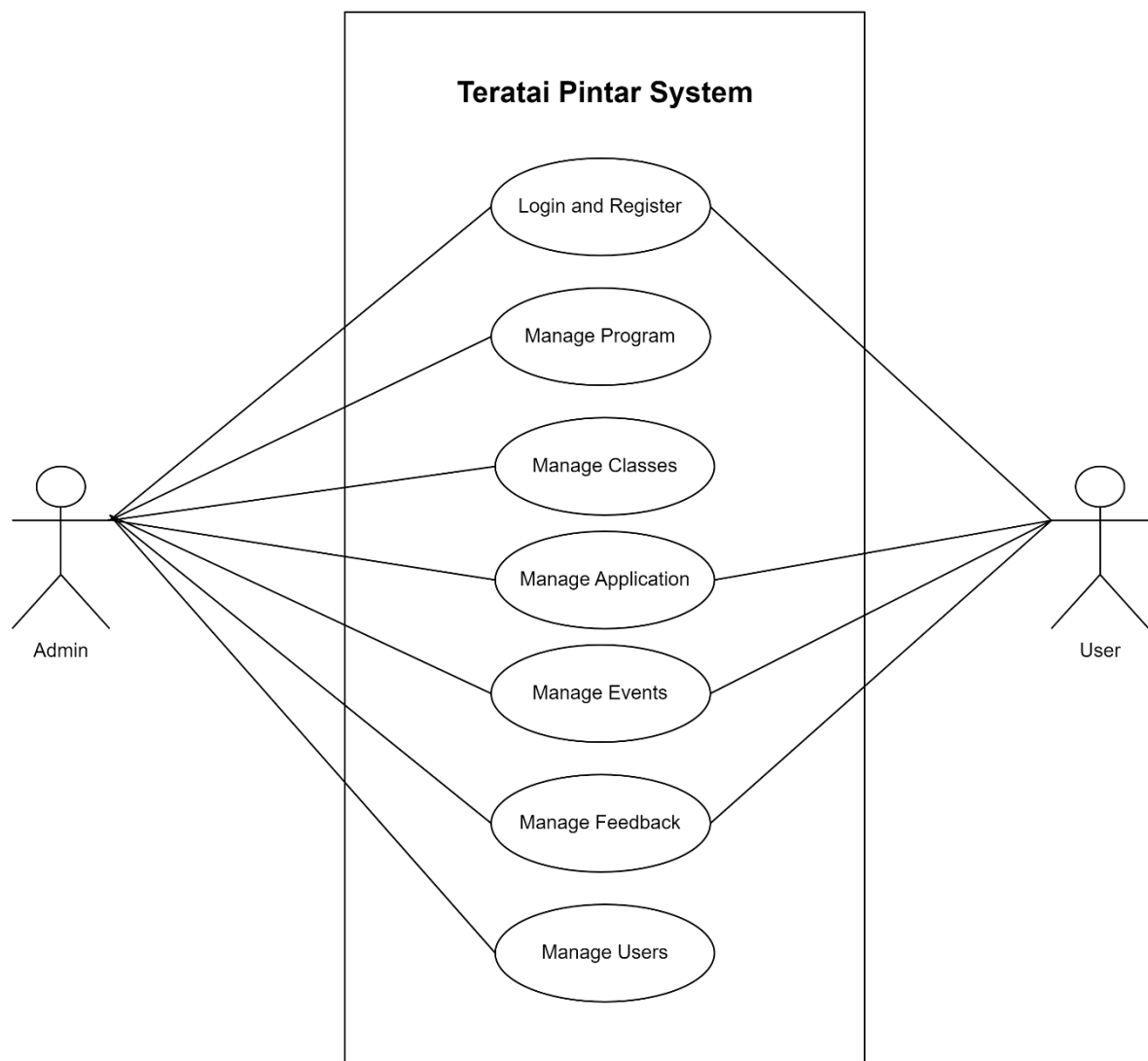


Figure 3. 1 Use Case Diagram

Module Explanation:

1. Login and Registration Module

- a. Facilitates secure logins for all authorized users with role-based access.

2. Manage Program Module

- a. Admin can perform operations related to managing educational programs.

Activities:

- Add new programs.
- Update program details.
- Deactivate or activate programs.
- View the list of all programs.

3. Manage Classes

- a. Admin can handle the management of classes programs.

Activities:

- Create new classes.
- Update class information.
- Delete the class.
- View list of class.

4. Manage Application

- a. Admin can access and review applications submitted by users.

Activities:

- View application details.
- accept or reject applications.

- b. Users can apply for enrollment.

Activities:

- Browse available programs and classes.
- Submit application details.
- Update application details.
- Receive confirmation or status updates.

5. Manage Event

- a. Admins organize and maintain information about various events.

Activities:

- Add new events.
- Update event details.
- View list of events.

- b. Users explore and view details about upcoming events.

Activities:

- Browse list of events.
- View event details and schedule.

6. Manage Feedback

- a. Admin can access and review feedback provided by users.

Activities:

- Create new classes.
- Update class information.
- Delete the class.
- View list of class.

- b. Users can offer feedback on various aspects of the system.

Activities:

- Provide comments or suggestions.

7. Manage Users

- a. Admin handle user management within the system

Activities:

- View list of users

3.2 UNIFIED MODELING LANGUAGE (UML) DIAGRAM

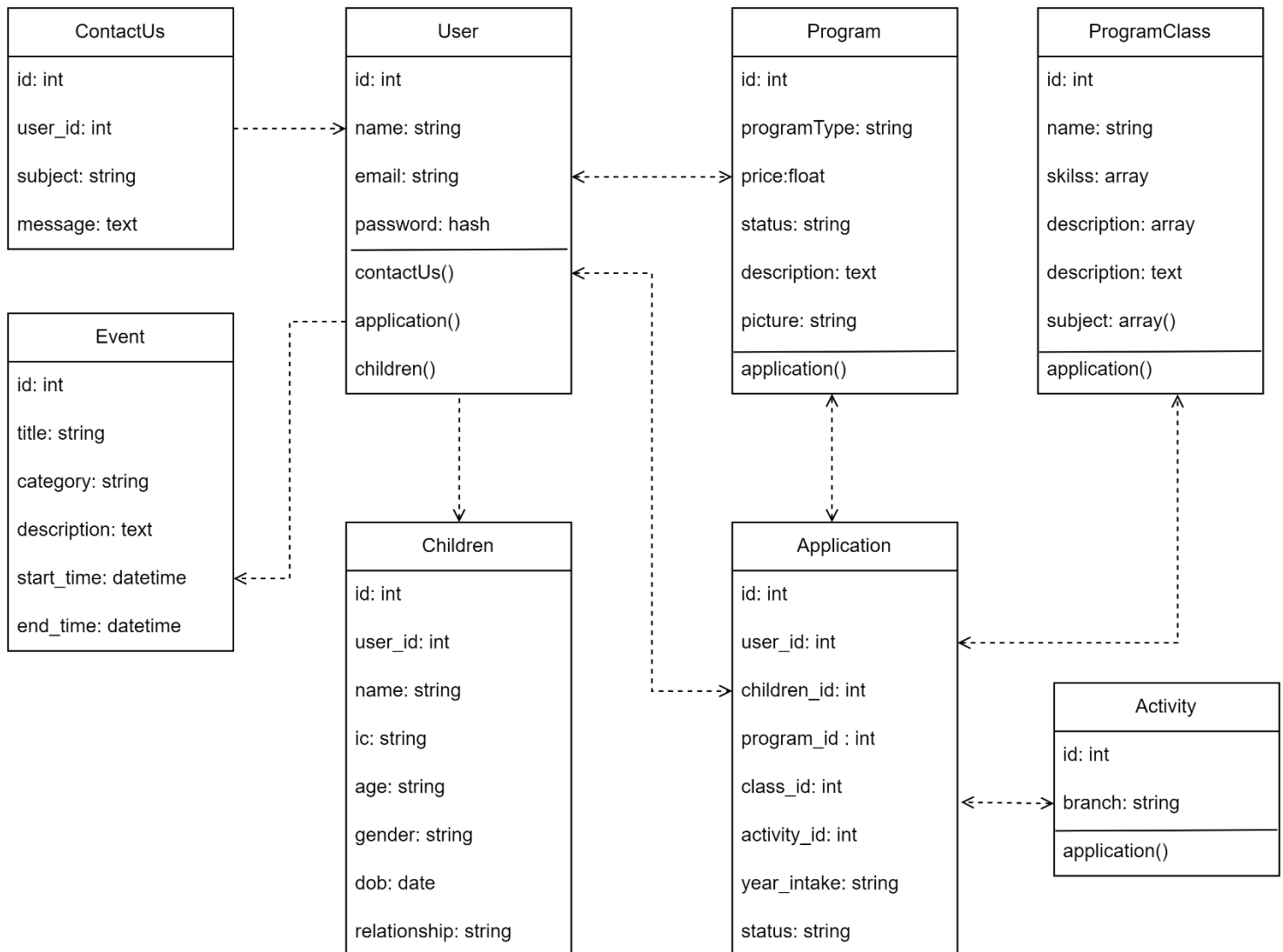


Figure 3. 2 UML Diagram

Explanation:

1. **User Class:**

- Represents a user in the system.
- Attributes: **id, name, email, password, role.**
- Methods: **contactUs(), application(), children()** to represent relationships.

2. **Program Class:**

- Represents a program offered in the system.
- Attributes: **id, programType, price, status, description, picture.**
- One-to-Many with Application (a program can have multiple applications).

3. **Application Class:**

- Represents an application made by a user for a program.
- Attributes: **id, user_id, program_id, class_id, activity_id, year_intake, status.**
- Relationship: Many-to-One with programClass, Activity, Program, User

5. **Children Class:**

- Represents information about the children of a user.
- Attributes: **id, user_id, name, ic, age, gender, dob, relationship.**
- Relationship: Many-to-One with User (a user can have multiple children).

6. **ContactUs Class:**

- Represents user-generated contact messages.
- Attributes: **id, user_id, subject, message.**
- Relationship: Many-to-One with User (a user can create multiple contact messages).

7. Event Class:

- Represents events in the system.
- Attributes: **id, title, category, description, start_time, end_time.**

8. ProgramClass Class:

- Represents classes related to a program.
- Attributes: **id, name, skills, description, subject.**
- Relationship: One-to-Many with Application (a program class can have multiple applications).

9. Activity Class:

- Represents different branch location in the system..
- Attributes: **id, branch**
- Relationship: One-to-Many with Application (a branch location can have multiple applications).

3.3 CONTROLLER FUNCTION

```
<?php

namespace App\Http\Controllers;

use App\Models\User;
use App\Models\Children;
use App\Models\Program;
use App\Models\ProgramClass;
use App\Models\Activity;
use App\Models\Application;
use Illuminate\Http\Request;

class ApplicationController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $user = User::where('id', auth()->user()->id)->first();
        $applications = Application::whereHas('user', function ($query) {
            $query->where('user_id', auth()->user()->id);
        }->paginate(10);

        return view('user.application.index', compact('user', 'applications'));
    }

    public function indexApp()
    {
        $user = User::where('id', auth()->user()->id)->first();
        $applications = Application::all();
        return view('admin.application.index', compact('applications', 'user'));
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        $user = User::where('id', auth()->user()->id)->first();
        $childrens = Children::All();
        $programs = Program::All();
        $classes = ProgramClass::All();
        $activities = Activity::All();
        return view('user.application.create', compact('user', 'childrens', 'programs', 'classes',
'activities'));
    }
}
```



```
}

/**
 * Store a newly created resource in storage.
 */
public function store(Request $request)
{
    $user = User::find($request->input('user_id'));
    $child = $user->children()->first();
    $program = Program::find($request->input('program_id'));
    $classes = ProgramClass::find($request->input('class_id'));
    $activities = Activity::find($request->input('activity_id'));
    $child = new Children();
    $child->name = $request->input('name');
    $child->ic = $request->input('ic');
    $child->age = $request->input('age');
    $child->gender = $request->input('gender');
    $child->dob = $request->input('dob');
    $child->relationship = $request->input('relationship');
    $child->save();
    $user->children()->save($child);
    $request->merge([
        'user_id' => auth()->user()->id,
        'program_id' => $program->id,
        'class_id' => $classes->id,
        'activity_id' => $activities->id,
    ]);
    Application::create($request->all());
    return redirect()->route('application.index')->with('success', 'Application Submitted!');
}

/**
 * Display the specified resource.
 */
public function show($id)
{
    $application = Application::find($id);
```

Figure 3. 3 ApplicationController

'ApplicationController' are specifically managing user applications for program enrollment. The structure is:

| Function | Explanations |
|--|--|
| index() | Retrieves applications for the currently authenticated user and renders the user's application index view. |
| indexApp() | Renders the admin view with all applications. |
| create() | Fetches necessary data (user details, children, programs, classes, and activities) and renders the application creation form. |
| store() | Handles the creation of a new application. It involves creating a new child entry, associating it with the user, and saving the application details. Redirects to the application index view with a success message. |
| show(\$id | Displays the details of a user's application. |
| showApp(\$id) | Displays the details of an application for admin. |
| edit(\$id) | Retrieves an application for editing along with the available programs, classes, and activities |
| update(Request \$request, Application \$application) | Updates the user's application based on the submitted form data. |
| destroy(\$id) | Deletes a user's application and redirects to the user's dashboard. |

```
<?php

namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;
use App\Models\ContactUs;

class ContactUsController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $user = User::where('id', auth()->user()->id)->first();
        $contactUs = ContactUs::all();
        return view('admin.contactUs.index', compact('contactUs', 'user'));
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        $user = User::where('id', auth()->user()->id)->first();
        // dd($users);
        return view('user.contactUs.index', compact('user'));
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        $user = User::find($request->input('user_id'));
        $request->merge([
            'user_id' => auth()->user()->id,
        ]);

        ContactUs::create($request->all());
        return redirect()->route('contactUs.create')->with('success', 'Your message has been sent. Thank you!');
    }
}
```

Figure 3. 4 ContactUsController

‘ContactUsController’ for managing user messages or inquiries through a contact form.

The structure are:

| Function | Explanations |
|--|---|
| index() | Retrieves all contact messages and renders the admin view for managing these messages. The view includes user details for context. |
| create() | Renders the user view for the contact form, allowing users to submit messages. |
| store() | Handles the storage of a new contact message. It associates the message with the authenticated user, creates a new entry in the ContactUs model, and redirects the user to the contact form with a success message. |
| show(\$id | Displays the details of a user's application. |
| showApp(\$id) | Displays the details of an application for admin. |
| edit(\$id) | Retrieves an application for editing along with the available programs, classes, and activities |
| update(Request \$request, Application \$application) | Updates the user's application based on the submitted form data. |
| destroy(\$id) | Deletes a user's application and redirects to the user's dashboard. |

```
<?php

namespace App\Http\Controllers;
use App\Models\User;
use App\Models\Event;
use Illuminate\Http\Request;

class EventController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $user = User::where(auth()->user()->role == '1');
        $events = Event::All();
        return view('admin.event.index', compact('events'));
    }

    public function display()
    {
        $events = Event::All();
        return view('user.event.index', compact('events'));
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        Event::create($request->all());
        return redirect()->route('event.index')->with('success', 'Event Successfully Created!');
    }

    /**
     * Show the form for editing the specified resource.
     */
    public function edit($id)
    {
        $event = Event::find($id);
        return view('event.index', compact('event'));
    }

    /**
     * Update the specified resource in storage.
     */
    public function update(Request $request, Event $event)
    {

```

```

        $event->start_time = $request->start_time;
        $event->end_time = $request->end_time;
        $event->save();
        return redirect()->route('event.index')->with('success', ' Event Program Successfully
Updated!');
    }
}

```

Figure 3. 5 EventController

‘EventController’ are for managing events within a system. The structure are:

| Function | Explanations |
|--|--|
| index() | Retrieves all events and renders the admin view for managing these events. The view is designed for administrators, considering the condition <code>auth()->user()->role == '1'</code> |
| display() | Handles the storage of a new event. It creates a new entry in the Event model using the submitted form data and redirects to the admin event index view with a success message. |
| edit(\$id) | Retrieves an event for editing based on the provided ID and renders the event index view for further actions. |
| update(Request \$request, Event \$event) | Updates the start and end times of a specified event based on the submitted form data. It then redirects to the admin event index view with a success message. |

```
<?php

namespace App\Http\Controllers;
use App\Models\Program;
use App\Models\ProgramClass;
use App\Models>ContactUs;
use App\Models\User;
use App\Models\Children;
use App\Models\Application;
use Illuminate\Http\Request;

class HomeController extends Controller
{
    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function index()
    {
        $programs = Program::All();
        $classes = ProgramClass::All();
        return view('user.dashboard',compact('programs','classes'));
    }
    public function indexAdmin()
    {
        $child = Children::All();
        $countApplication = Application::count();
        $countProgram = Program::count();
        $countClass = ProgramClass::count();
        $countFeedback = ContactUs::count();
        return
view('admin.dashboard',compact('countApplication','countProgram','countClass','countFeedback','child')
);
    }

    public function program()
    {
        $programs = Program::All();
        return view('user.program.index',compact('programs'));
    }
}
```

```

public function class()
{
    $classes = ProgramClass::All();
    return view('user.class.index',compact('classes'));
}

public function showProgram($id)
{
    $programs = Program::find($id);
    return view('user.program.show',compact('programs'));
}

public function showClass($id)
{
    $classes = ProgramClass::find($id);
    return view('user.class.index',compact('classes'));
}

public function user()
{
    $users = User::All();
    return view('admin.user.index',compact('users'));
}
}

```

Figure 3. 6 HomeController

‘HomeController’ are for managing various functionalities related to dashboard, programs, classes, and user administration. The structure are:

| Function | Explanations |
|-----------------|--|
| __construct | method ensures that only authenticated users can access the functions within this controller using the 'auth' middleware. |
| index() | Renders the user dashboard view, providing data on available programs and classes. |
| indexAdmin() | Renders the admin dashboard view, displaying counts of applications, programs, classes, and feedback, as well as information about children. |
| program() | Displays detailed information about a specific program based on the provided ID. |
| showClass(\$id) | Displays detailed information about a specific class based on the provided ID. |
| user() | Renders the admin view displaying all registered users. |


```
<?php

namespace App\Http\Controllers;
use App\Models\User;
// use App\Models\Program;
use App\Models\ProgramClass;
use Illuminate\Http\Request;

class ProgramClassController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $user = User::where(auth()->user()->role == '1');
        // $programs = Program::All();
        $classes = ProgramClass::All();
        return view('admin.class.index',compact('classes'));
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        $user = User::where(auth()->user()->role == '1');
        // $programs = Program::All();
        return view('admin.class.create');
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        ProgramClass::create($request->all());
        return redirect()->route('class.index')->with('success', 'Subject Created!');
    }

    /**
     * Display the specified resource.
     */
    public function show($id)
    {
        $classes = ProgramClass::find($id);
        return view('admin.class.show',compact('classes'));
    }
}
```

```

    }

    /**
     * Remove the specified resource from storage.
     */
    public function destroy($id)
    {
        $classes = ProgramClass::find($id);
        $classes->delete();
        return redirect()->route('class.index')->with('success', 'Class Deleted Successfully!');
    }
}

```

Figure 3. 7 ProgramClassController

‘ProgramClassController’ are for managing program classes within system. The structure:

| Function | Explanations |
|--------------------------|--|
| index() | Retrieves all program classes and renders the admin view for managing these classes. The view is designed for administrators, considering the condition <code>auth()->user()->role == '1'</code> |
| create() | Renders the admin view for creating a new program class. |
| store(Request \$request) | Handles the storage of a new program class. It creates a new entry in the ProgramClass model using the submitted form data and redirects to the admin class index view with a success message. |
| show(\$id) | Displays detailed information about a specific program class based on the provided ID. |
| destroy(\$id) | Deletes a program class based on the provided ID and redirects to the admin class index view with a success message. |

```
<?php

namespace App\Http\Controllers;
use App\Models\User;
use App\Models\Program;
use Illuminate\Http\Request;

class ProgramController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $user = User::where(auth()->user()->role == '1');
        $programs = Program::All();
        return view('admin.program.index',compact('programs'));
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        $user = User::where(auth()->user()->role == '1');
        return view('admin.program.create');
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        $request->validate([
            'image' => 'required|image|mimes:jpeg,png,jpg,gif|max:2048',
        ]);
        $picturePath = $request->file('image')->store('public/banner');

        Program::create([
            'programType' => $request->programType,
            'price' => $request->price,
            'status' => 'Active',
            'description' => $request->description,
            'picture' => $picturePath,
        ]);

        return redirect()->route('program.index')->with('success', 'Program Created!');
```

```
}

/**
 * Display the specified resource.
 */
public function show($id)
{
    $program = Program::find($id);
    return view('admin.program.show',compact('program'));
}

/**
 * Show the form for editing the specified resource.
 */
public function edit($id)
{
    $program = Program::find($id);
    return view('program.index', compact('program'));
}

/**
 * Update the specified resource in storage.
 */
public function update(Request $request, Program $program)
{
    $program->status = $request->status;
    $program->save();
    return redirect()->route('program.index')->with('success', ' Status Program Successfully
Updated!');
}

/**
 * Remove the specified resource from storage.
 */
public function destroy($id)
{
    $program = Program::find($id);
    $program->delete();
    return redirect()->route('program.index')->with('success', 'Program Deleted Successfully!');
}
}
```

Figure 3. 8 ProgramController

‘ProgramController’ are for managing programs within a system. The structure is:

| Function | Explanations |
|--|--|
| index() | Retrieves all programs and renders the admin view for managing these programs. The view is designed for administrators, considering the condition <code>auth()->user()->role == '1'</code> . |
| create() | Renders the admin view for creating a new program. |
| store(Request \$request) | Handles the storage of a new program. It validates the incoming request for an image, stores the image in the 'public/banner' directory, and creates a new entry in the Program model using the submitted form data. Redirects to the admin program index view with a success message. |
| update(Request \$request, Program \$program) | Updates the status of a specified program based on the submitted form data. It then redirects to the admin program index view with a success message. |
| destroy(\$id) | Deletes a program based on the provided ID and redirects to the admin program index view with a success message. |

3.4 OBJECT RELATIONAL MAPPING (ORM)

Object-Relational Mapping (ORM) is a technique used to interact with a database using an object-oriented approach. In Laravel, Eloquent is the ORM used to simplify database interactions by representing database tables as classes and table rows as objects.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Application extends Model
{
    use HasFactory;
    protected $fillable = [
        'user_id',
        'children_id',
        'program_id',
        'class_id',
        'activity_id',
        'year_intake',
        'status',
    ];

    public function user()
    {
        return $this->belongsTo(User::class);
    }

    public function program()
    {
        return $this->belongsTo(Program::class);
    }

    public function activity()
    {
        return $this->belongsTo(Activity::class);
    }

    public function programClass()
    {
        return $this->belongsTo(ProgramClass::class, 'class_id');
    }
}
```

Figure 3. 9 Application Model

Application model is associated with a database table, and it has the following fillable fields:

| Fillable Fields | Explanations |
|-----------------|---|
| user_id | Represents the user associated with the application. |
| children_id | Represents the child associated with the application. |
| program_id | Represents the program selected in the application. |
| class_id | Represents the program class selected in the application. |
| activity_id | Represents the activity selected in the application. |

Therefore, the relationship is define :

| Relationship | Explanations |
|----------------|--|
| user() | Defines a belongs-to relationship with the User model, indicating that an application belongs to a user. |
| program() | Defines a belongs-to relationship with the Program model, indicating that an application belongs to a program. |
| activity() | Defines a belongs-to relationship with the Activity model, indicating that an application belongs to an activity. |
| programClass() | Defines a belongs-to relationship with the ProgramClass model using the class_id foreign key, indicating that an application belongs to a program class. |

```

8      @method('PUT')
9      <div class="col-md-6">
10         <div class="form-floating">
11             <select class="form-select" id="floatingSelect" aria-label="ProgramType" name="program_id">
12                 <option selected disabled>{{ $application->program->programType }}</option>
13                 @foreach ($programs as $program)
14                     <option value="{{ $program->id }}">{{ $program->programType }}</option>
15                 @endforeach
16             </select>
17             <label for="floatingSelect">Program Type</label>
18         </div>
19     </div>
20     <div class="col-md-6">
21         <div class="form-floating">
22             <select class="form-select" id="floatingSelect" aria-label="name" name="class_id">
23                 <option selected disabled>{{ $application->programClass->name }}</option>
24                 @foreach ($classes as $class)
25                     <option value="{{ $class->id }}">{{ $class->name }}</option>
26                 @endforeach
27             </select>
28             <label for="floatingSelect">Class</label>
29         </div>
30     </div>

```

Figure 3. 10 Blade file for edit application

This Laravel Blade view represents the edit application page, allowing users to modify details of an application. It showcases how Eloquent data retrieved from the Application model is presented to users, displaying relevant order information fetched from the database in a structured format. In Laravel's Eloquent ORM, nested relationships allow to access related data through chained relationships. For example:

\$application->user->children->first()->name: Here, user represents the relationship with the User model, children represents the relationship with the Children model, and name is an attribute of the Children model. This chain of relationships allows accessing the name of the child associated with the application.

\$application->user: This represents a relationship between the Application model and the User model. In Eloquent, relationships are defined using methods in the model. In this case, the user() method is defined in the Application model. When accessed, it returns the related User model.

\$application->user->email: Accessing the email attribute of the User model associated with the application.

\$application->user->children->first()->relationship: Accessing the relationship attribute of the Children model associated with the user.

\$application->programClass-name: Accessing the name attribute of the programClass model associated with the application.

The screenshot shows the 'View application' page of the Teratai Pintar web application. The page has a light blue header with the Teratai Pintar logo, a search bar, and navigation links: Home, Programme, Class, Apply, Event, Contact Us, and User. The main content area is titled 'View application' with a breadcrumb 'Home / Application / View'. Below this is a 'Details of Application' section. It is divided into three parts: 1.0 School & Programme, 2.0 Children Details, and 3.0 Personal Details. Each part contains a form with input fields for various attributes.

| 1.0 School & Programme | |
|--------------------------------------|--------------------------------------|
| Program Type Preschool (Full Day) | Class Teratai Kids Islamic |
| Year Intake 2024 | Branch Teratai Pintar, Setia Alam |

| 2.0 Children Details | |
|------------------------------------|-------------------|
| Children Name Syazwani Nadhirah | |
| IC 010530101212 | Age 4 |
| Gender Female | DOB 2024-01-21 |

| 3.0 Personal Details | |
|------------------------|---------------------------|
| Name User | Email user@example.com |
| Relationship Mother | |

Back

Figure 3. 11 view page

This view represents the live interaction of the Laravel application with the database through Eloquent ORM. It visualizes the data retrieved from the Application model and presented to users in the browser interface.

3.5 VIEW ARRANGEMENT AND LAYOUT STRUCTURE

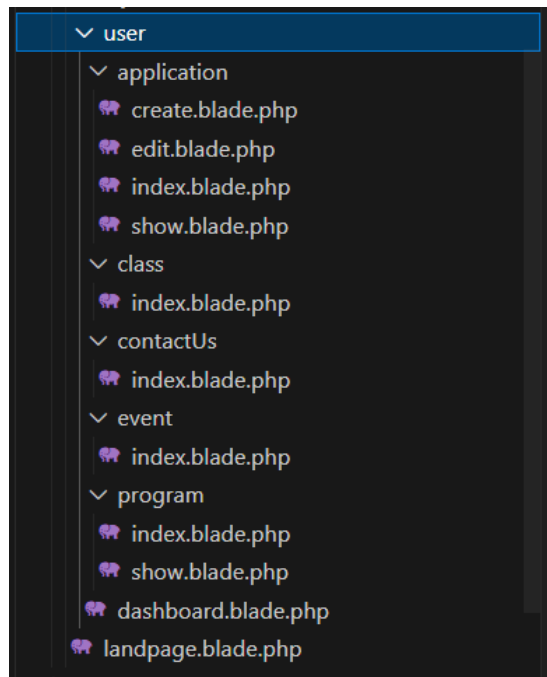


Figure 3. 12 User Folder

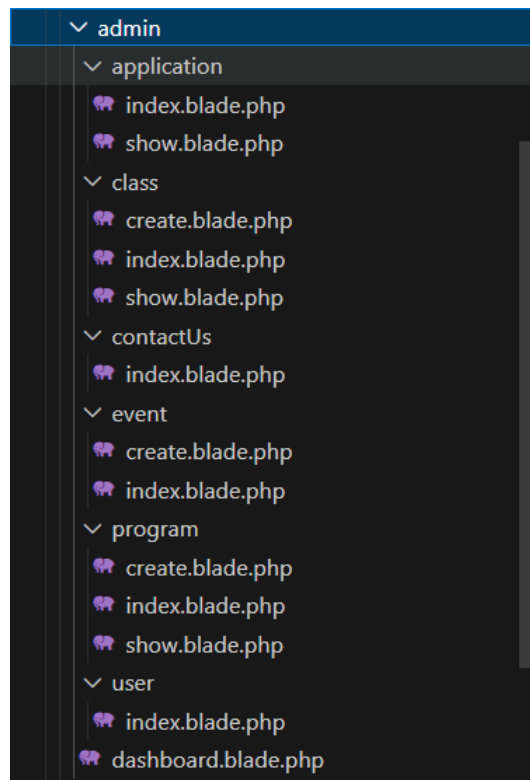


Figure 3. 13 Admin Folder

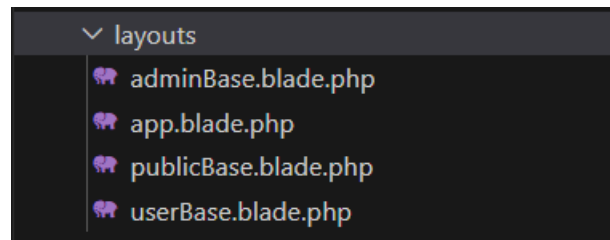


Figure 3. 14 layouts folder

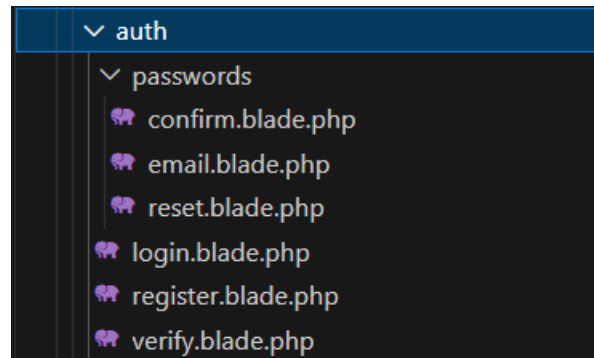


Figure 3. 15Auth Folder

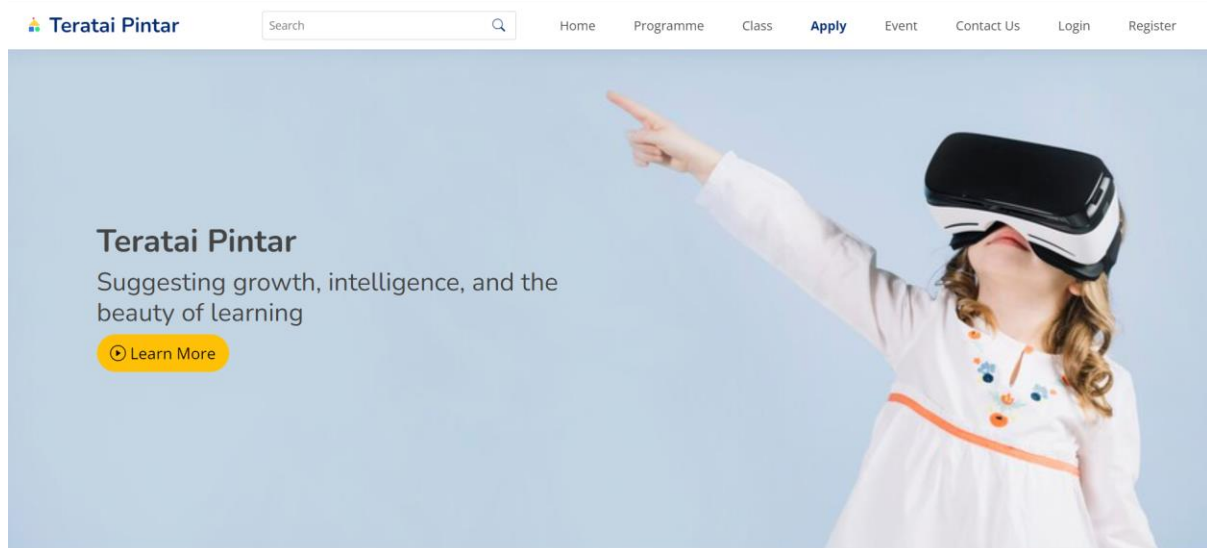


Figure 3. 16 Landpage for guest

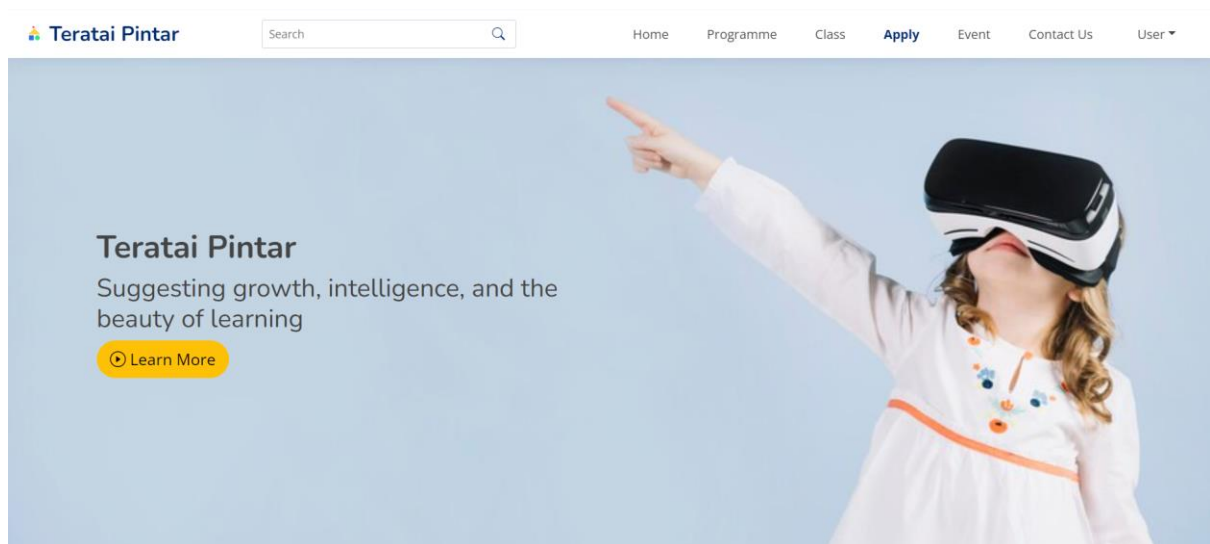


Figure 3. 17 user dashboard

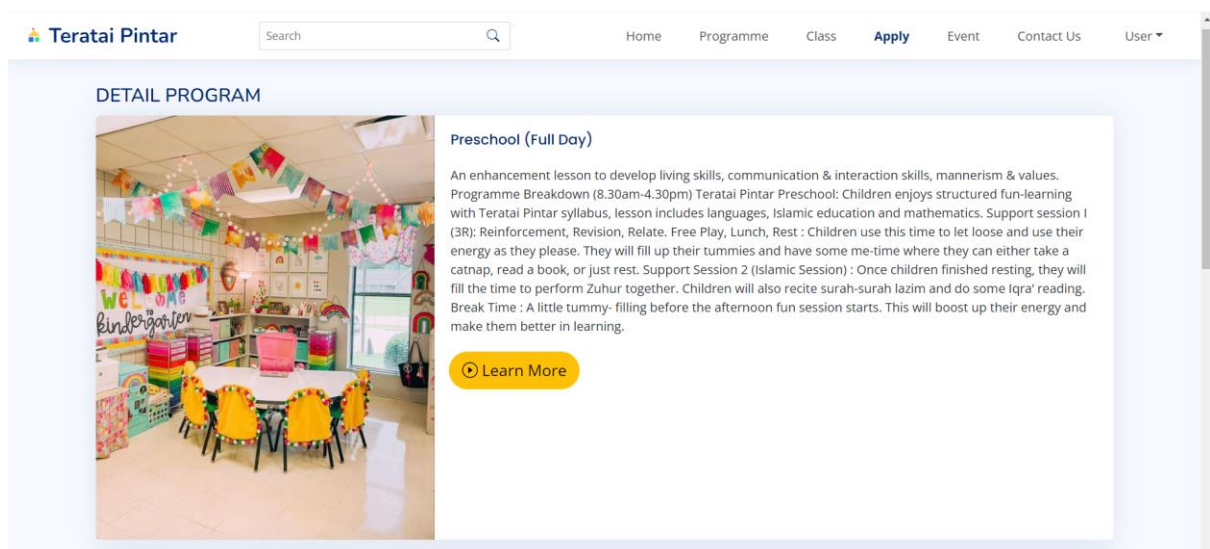


Figure 3. 18 User program dashboard

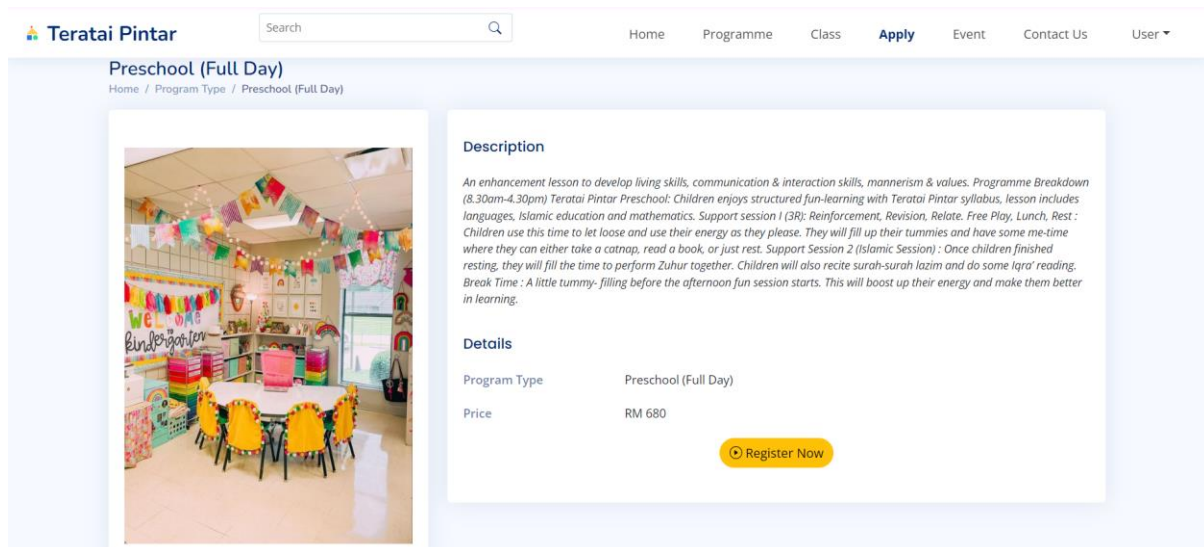


Figure 3. 19 User program display

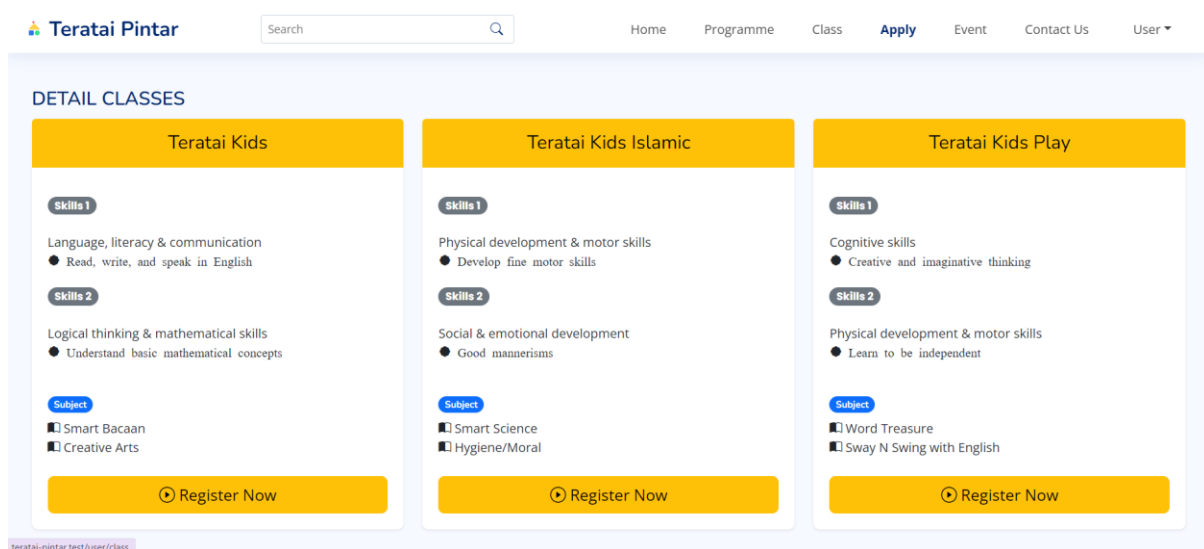


Figure 3. 20 User class dashboard

Teratai Pintar Web Application, UMPSA Pekan, 2023

Teratai Pintar Search Home Programme Class **Apply** Event Contact Us User

Create application

Home / Application / Create

Please Fill the Form

1.0 School & Programme

Program Type Please Select

Class Please Select

Year Intake Please Select

Branch Please Select

2.0 Children Details

Children Name

IC

Age

Gender Please Select

DOB dd/mm/yyyy

Figure 3. 21 user application form

Teratai Pintar Search Home Programme Class **Apply** Event Contact Us User

List Application



| No | Program Type | Class | Intake | Branch | Created At | Status | Action |
|----|----------------------|----------------------|--------|----------------------------|-----------------|---------|---|
| 1 | Preschool (Full Day) | Teratai Kids Islamic | 2024 | Teratai Pintar, Setia Alam | 06 January 2024 | Pending |   |

Figure 3. 22 user application list

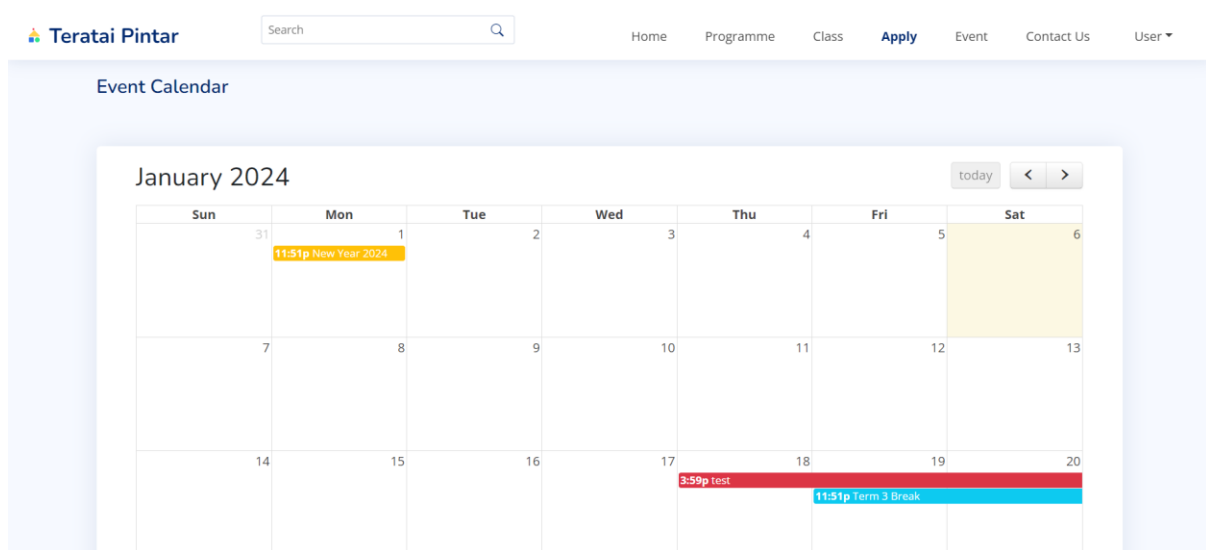


Figure 3. 23 User event dashboard

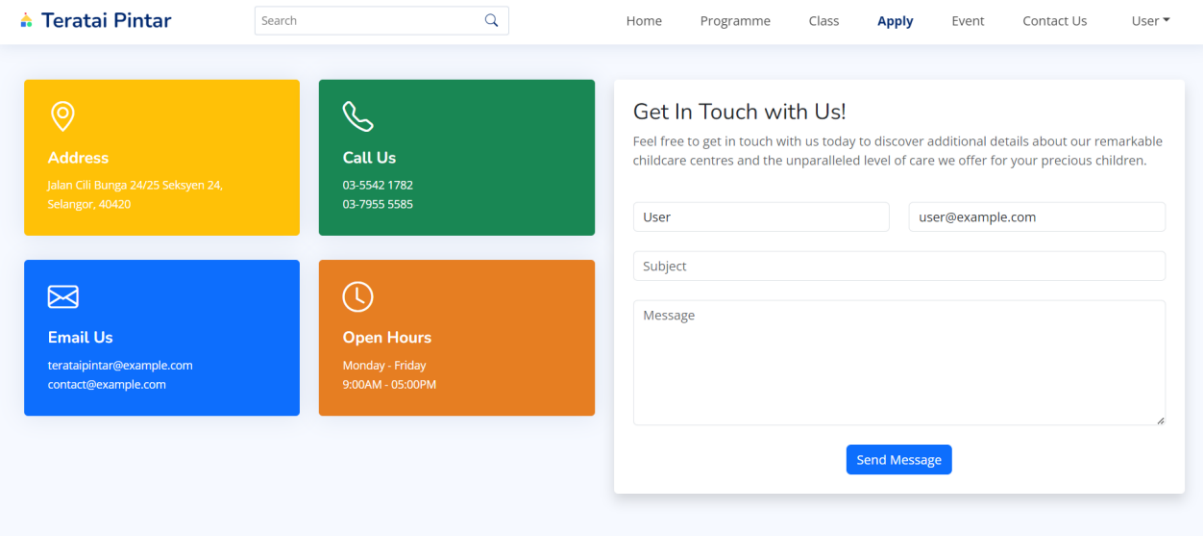


Figure 3. 24 User Contact Us

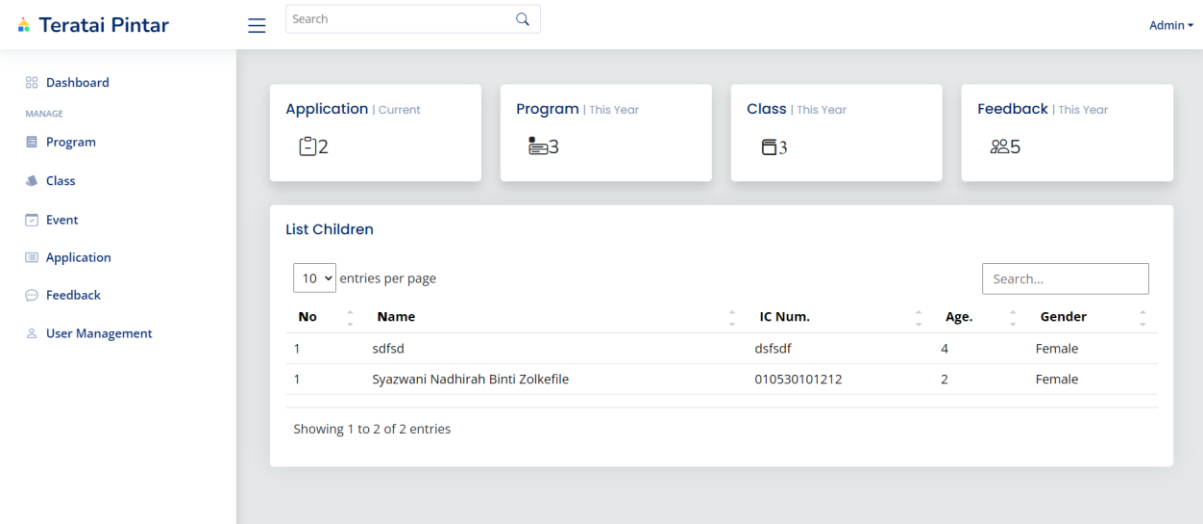


Figure 3. 25 Admin dashboard

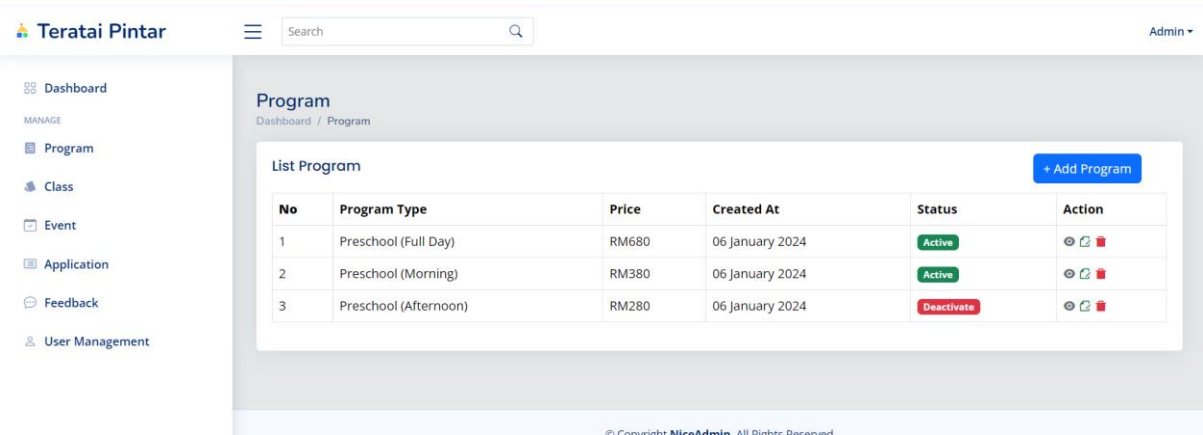


Figure 3. 26 Admin program dashboard

Teratai Pintar Search Admin

Create Program
Dashboard / Program / Create

Please Insert The Program.

ProgramType
Please Select

Price(RM)

Image
Choose File No file chosen

Description Program

Back Submit

Figure 3. 27 Admin program create

Teratai Pintar Search Admin

Detail Program
Dashboard / Program / View

Description

An enhancement lesson to develop living skills, communication & interaction skills, mannerism & values.
Programme Breakdown (8.30am-4.30pm) Teratai Pintar Preschool: Children enjoys structured fun-learning with Teratai Pintar syllabus, lesson includes languages, Islamic education and mathematics. Support session 1 (3R): Reinforcement, Revision, Relate. Free Play, Lunch, Rest : Children use this time to let loose and use their energy as they please. They will fill up their tummies and have some me-time where they can either take a catnap, read a book, or just rest. Support Session 2 (Islamic Session) : Once children finished resting, they will fill the time to perform Zuhur together. Children will also recite surah-surah lazim and do some Iqra' reading. Break Time : A little tummy- filling before the afternoon fun session starts. This will boost up their energy and make them better in learning.

Details

| | |
|--------------|----------------------|
| Program Type | Preschool (Full Day) |
| Price | RM 680 |

Back

Figure 3. 28 Admin program display

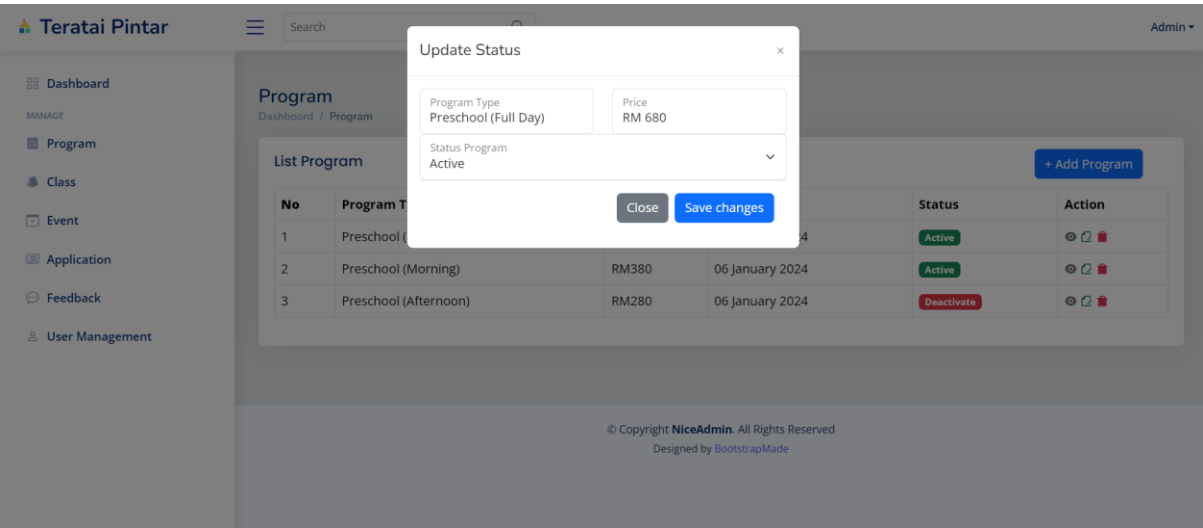


Figure 3. 29 Admin program update

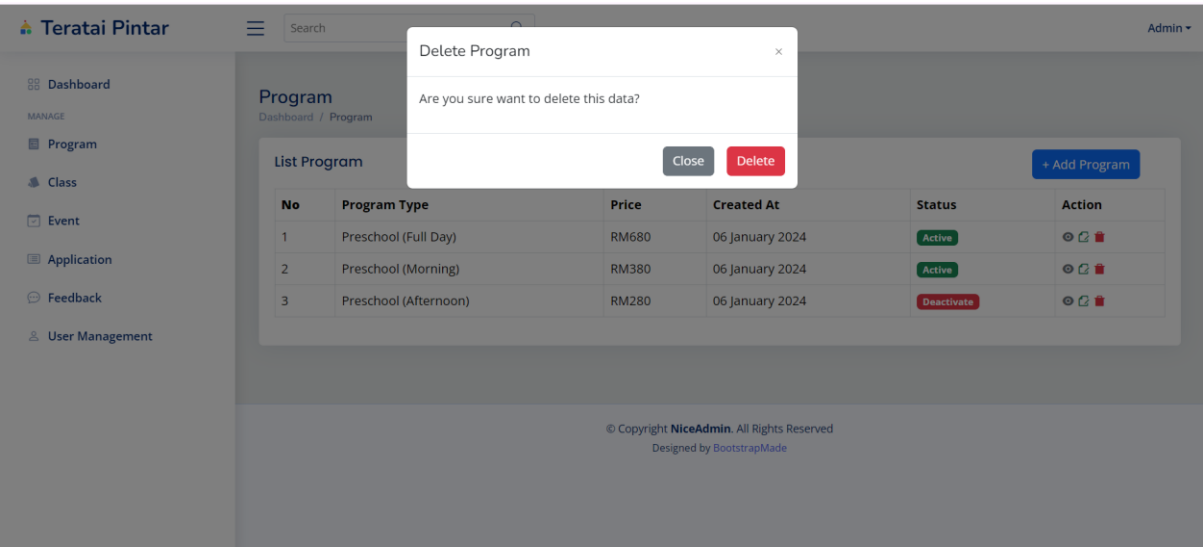


Figure 3. 30 Admin program delete

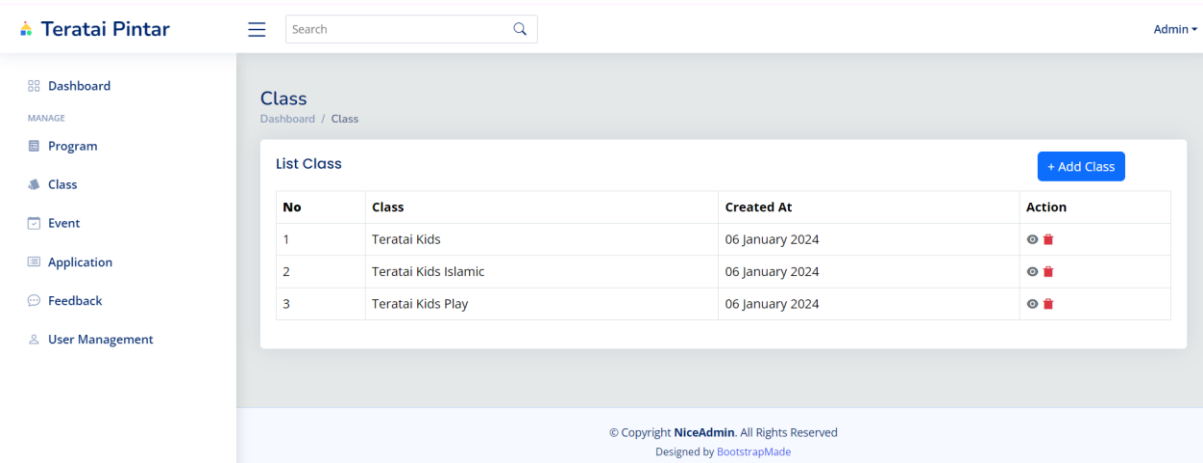


Figure 3. 31 Admin class dashboard

Teratai Pintar

Search

Admin

Dashboard

MANAGE

Program

Class

Event

Application

Feedback

User Management

Please Fill Form.

Class Name

Add Skills

Skills

Description

Add Skills

Add Subject

Subject

Add Subject

Back Submit

Figure 3. 32 Admin class create

Teratai Pintar

Search

Admin

Dashboard

MANAGE

Program

Class

Event

Application

Feedback

User Management

Class Details

Dashboard / Class / View

Teratai Kids

Class Details

| | |
|---------------|--|
| Skills 1 | Language, literacy & communication |
| Description 1 | Read, write, and speak in English |
| Skills 2 | Logical thinking & mathematical skills |
| Description 2 | Understand basic mathematical concepts |
| Subject1 | Smart Bacaan |
| Subject2 | Creative Arts |

Back

Figure 3. 33 Admin class view

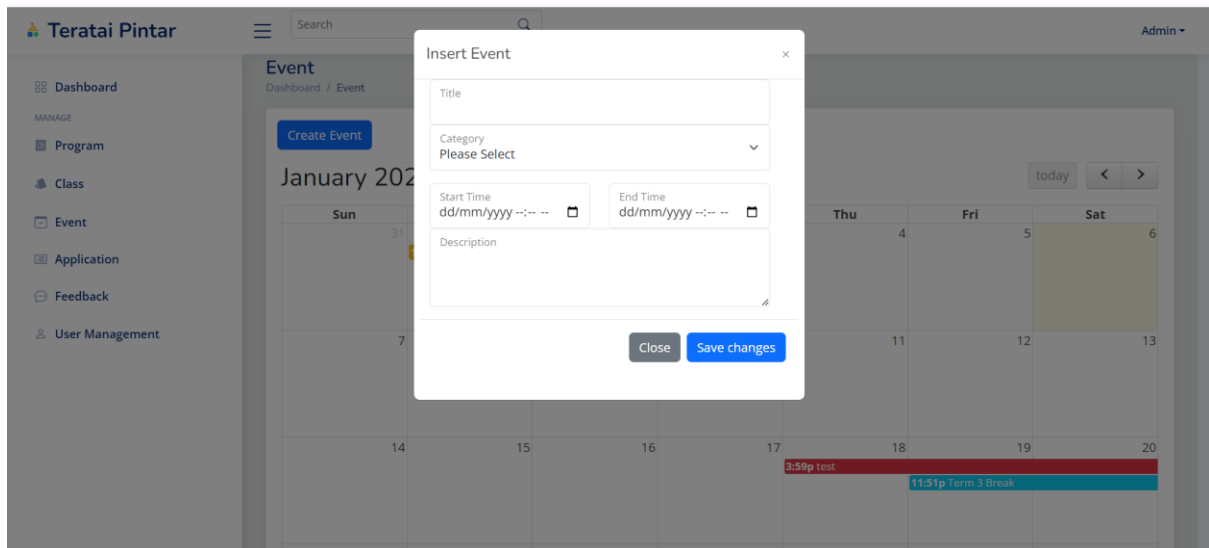


Figure 3. 34 Admin event create

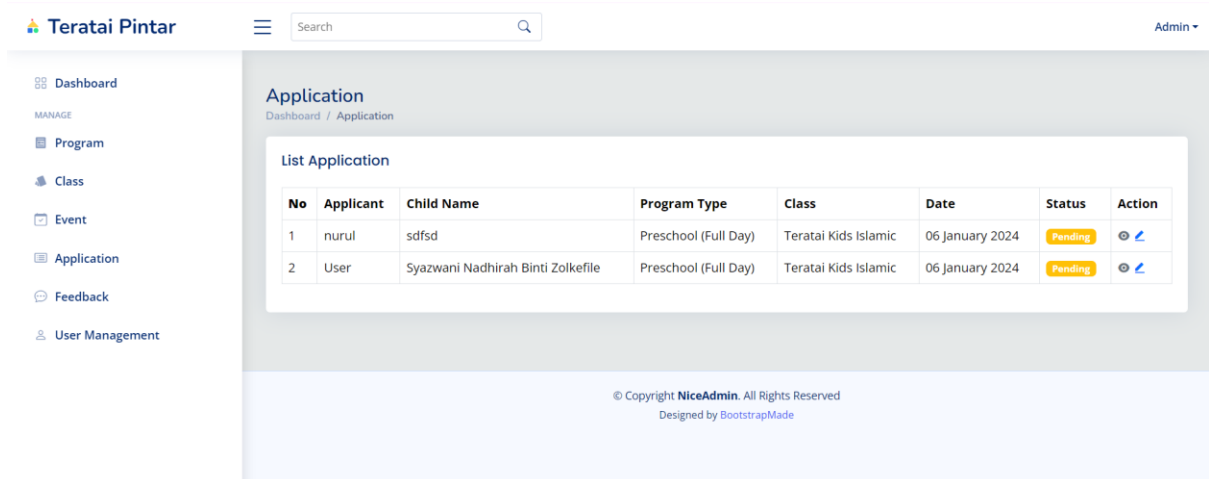


Figure 3. 35 Admin application dashboard

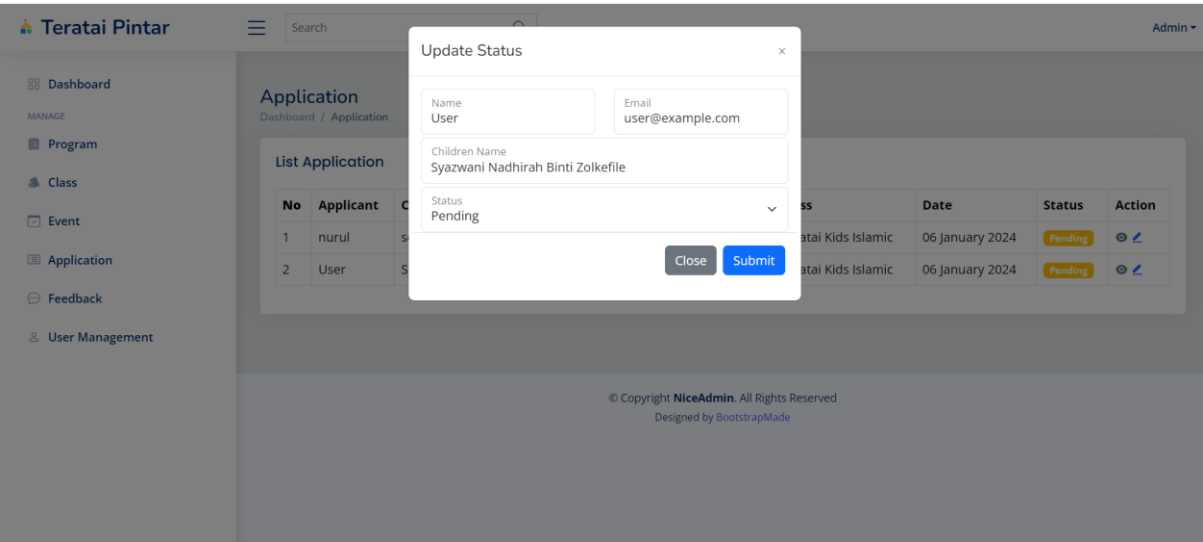


Figure 3. 36 Admin application update

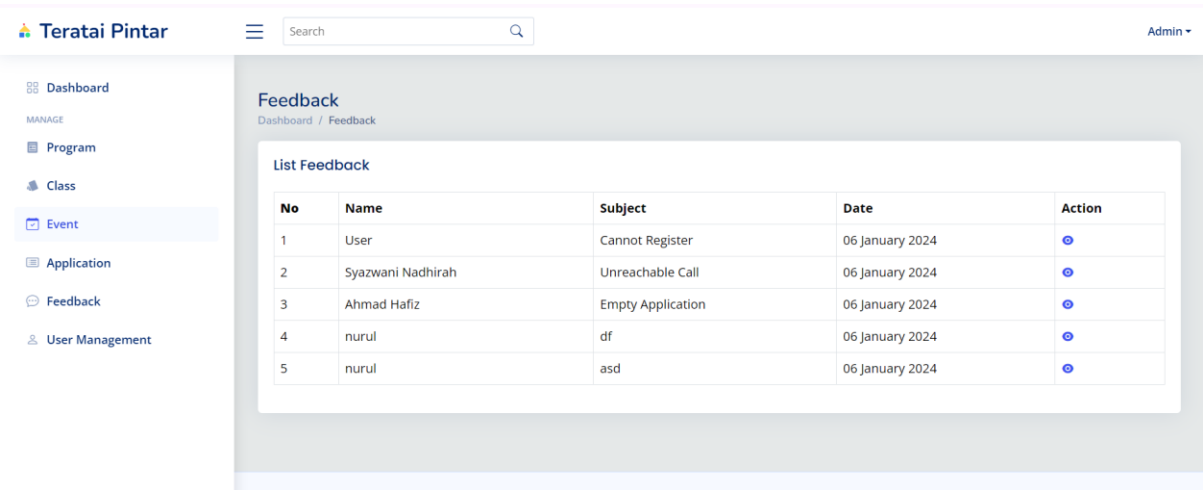


Figure 3. 37 Admin feedback dashboard

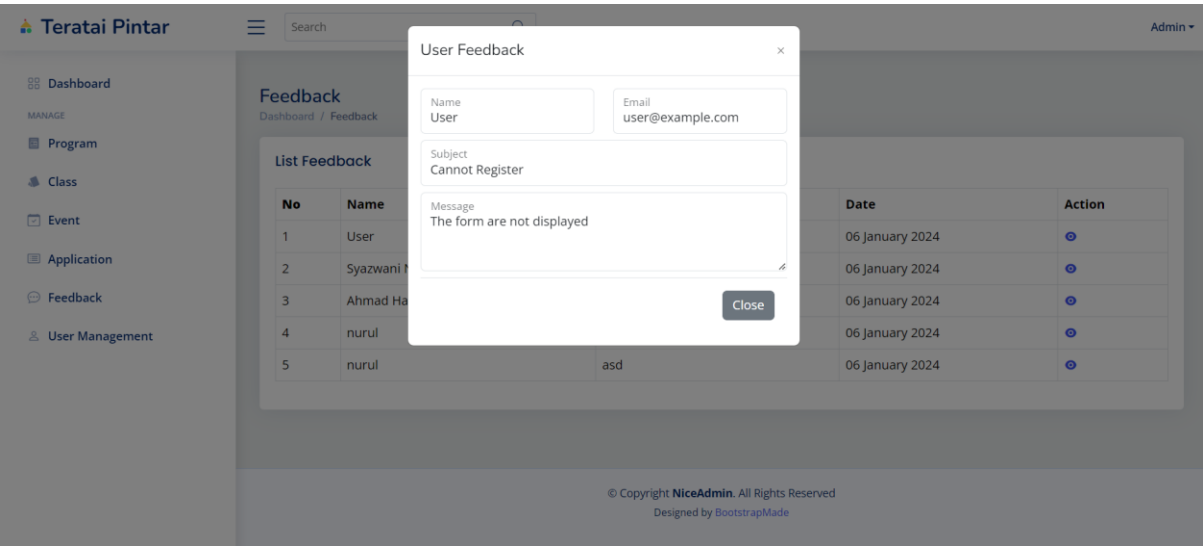


Figure 3. 38 Admin feedback view

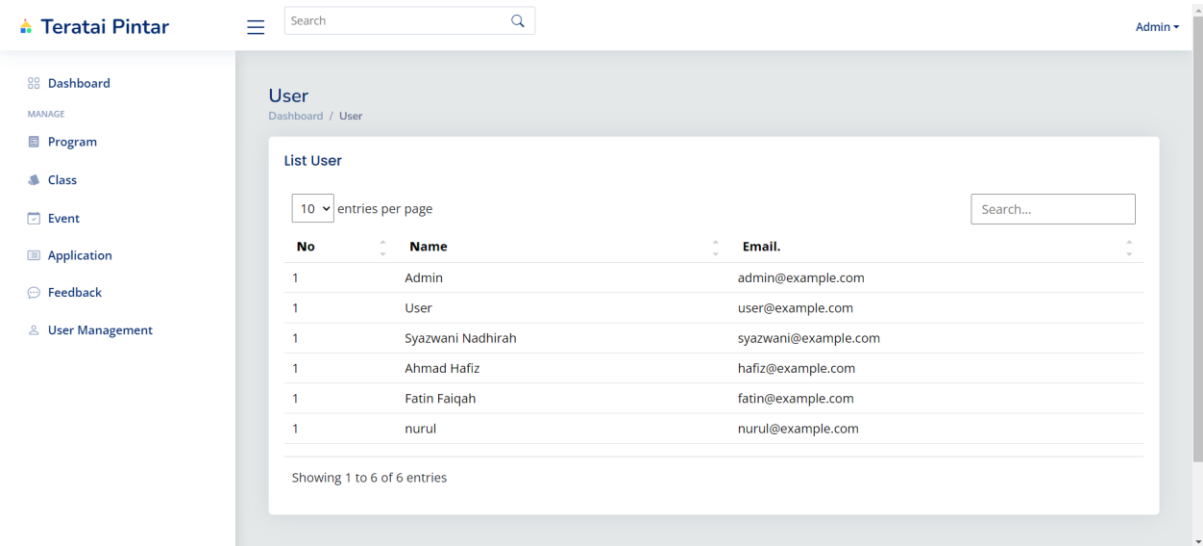


Figure 3. 39 Admin user dashboard

3.6 DEPLOYMENT IN INDAH SERVER

Url: [Teratai Pintar \(ump.edu.my\)](https://ump.edu.my/teratai-pintar/)

Drive link:

<https://drive.google.com/drive/folders/1Q2hh771JfZzfWnXalcKlxGBfLXbDz04x?usp=ssharing>

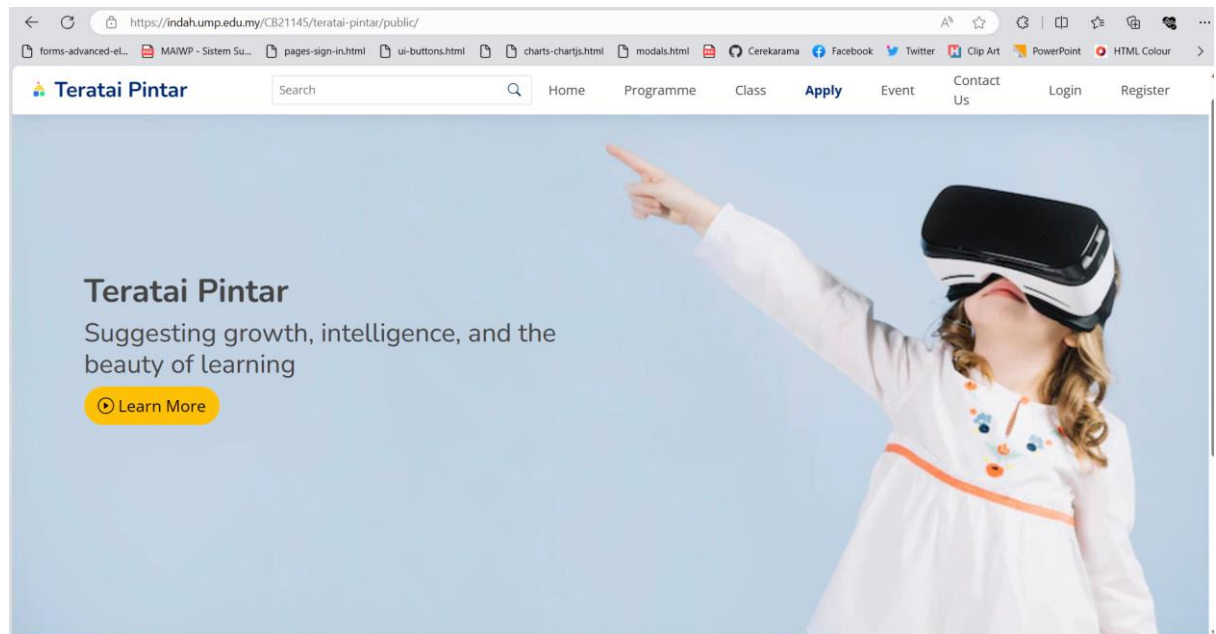


Figure 3. 40 Indah server deployment