# Department of Computing

# CS-344: Web Engineering

**Class:** BESE-12AB

**Group Members:**

Jazib (394006)  
Izzah Aslam (377547)

**Lab 11: Open House FYP Management System**

**Date: December 20, 2023**

**Instructor:** Sir Fahad Ahmed Satti

GitHub Repo Link: <https://github.com/Jazib-2004/Open-House-Management-Web-Assignment-3->

**Authorization**

Route::get('/', function () {

    return view('start');

});

Route::get('/login', function () {

    return view('login');

});

Route::post('/login',[AuthController::class,"login"]);

Route::get('/registeration', function () {

    return view('registeraton');

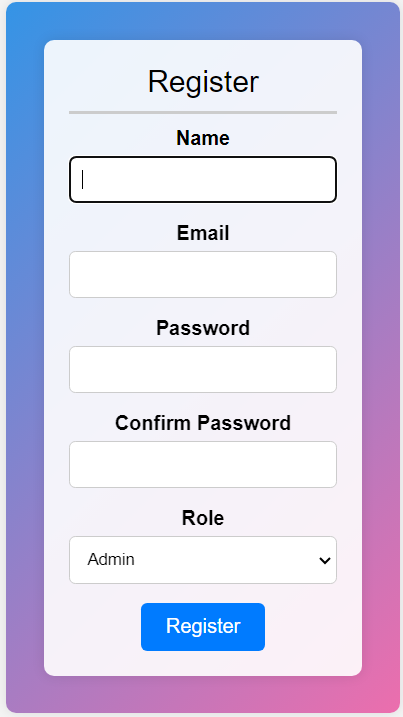
})->name('registeration');

Route::post('/registeration',[AuthController::class,"register"]);

**Registeration Page**

When we go to registeration page and the user enters all his details through post we pass it to controller class to provide logic what to do with it A screen shot of a computer

Description automatically generated

 First we see if the role is admin evaluator or the fyp\_group

Though switch case

For this our controller class should be linked with these tables thorugh mode and dabatase

use App\Models\Admin;

use App\Models\Fyp\_group;

use App\Models\Evaluator;

use Illuminate\Support\Facades

\DB;

Getting the submitted items thorugh request it gives to this controller

public function login(Request $request){

 $email = $request->input('email');

Checking the roles for example admin

case 'admin':

            try {

                $userId = DB::table("admins")->insertGetId([

                    "name" => $request->input('name'),

                    'email' => $request->input('email'),

                    'password' => bcrypt($request->input('password')),

                ]);

            } catch (\Exception $e) {

                Log::error('Error creating admin: ' . $e->getMessage());

            }

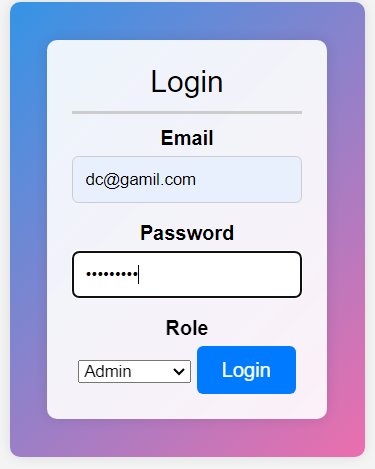
            break;

method of database class DB and using insertGetId whichwill give the id back to I just used it to check whether I am getting the right answer by writing dd($query) if it gives the id then my insertion was successful

Once the registeration is successful if the email is not found in the database of that role then we redirect to login page for signup with message

  return redirect('/login')->with('success', 'Registration successful. Please login.');

**Login Page:**

same as registeration page :

checking the role then through switch case go to that particular routine and then check whether the requested email is in that table or not

$user = Admin::where('email', $email)->first();

                if ($user){

then we de hash the password to match it with what we stored in our database

if(\Hash::check($password, $user->password)){

                    // Authentication successful

                    return redirect('/landing');

if it does then redirect to the landing page of that particular role

All this is done in AuthController class having methods login and register : as mentioned above

**Admin page:**

Admin can see all the projects with full details and define rurics for particular project and also select its location :

All these things are done in Detailscontroller class

Route::get("/landing",[DetailsController::class,"show\_project"]);

Route::post("/set\_location",[DetailsController::class,"setLocation"]);

Route::post("/add\_rubrics",[DetailsController::class,"addRubrics"]);

When admin login it comes to the landing page and show\_project method will be called

// Get all projects with their associated images

        $projects = Project::with('images')->get();

        // Get all available locations

        $locations = Location::all();

        return view('show\_project', ['projects' => $projects, 'locations' => $locations]);

It simply takes all the details from the project table and its associated images the function we defined in its model class

And returns the show\_project blade file.

<option value="" {{ $project->location\_id ? '' : 'selected' }}>Select a location</option>

If that project has associated location id then we place that location as the selected one on dropmenu

When the admin selects form is submitted to

   <form method="post" action="/set\_location">

For this route is already made above

if ($project->location\_id) {

            return redirect()->back()->with('error', 'Project already has a location assigned.');

        }

For this first we see if it already has a allocated location if not then find

$location = Location::find($locationId);

And delete from location table and insert that in taken location table

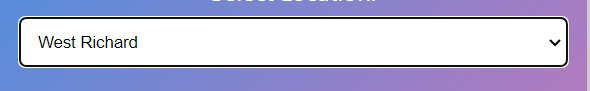
DB::table('taken\_locations')->insert([

            'name' => $location->name,

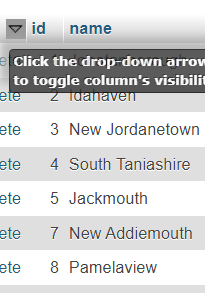
        ]);

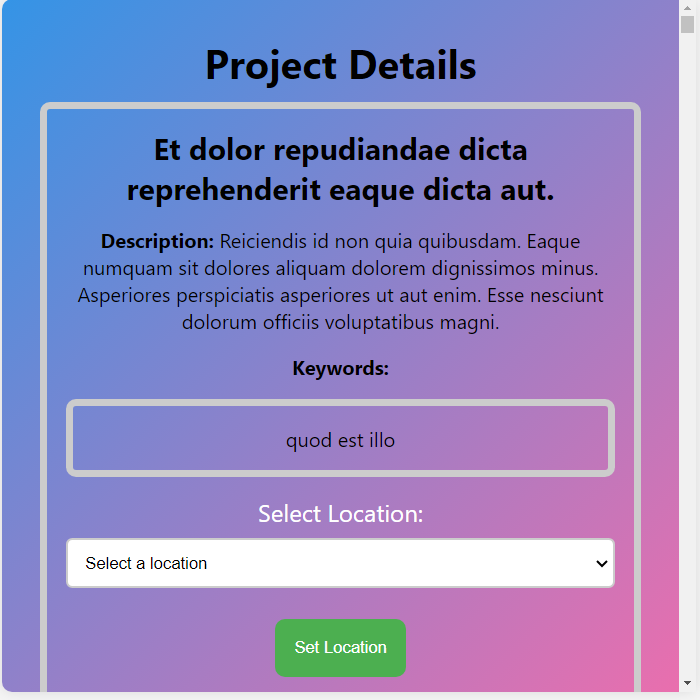
        $location->delete();

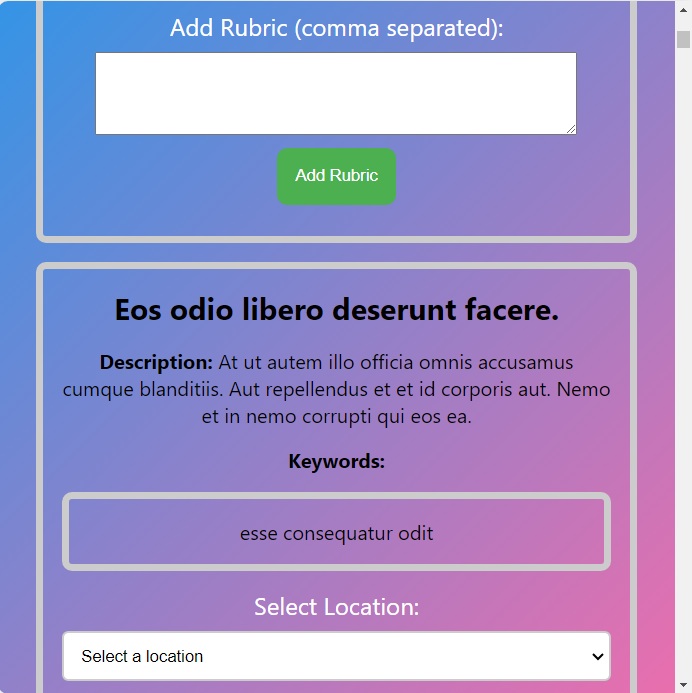
If we select it



After click on the button the request is given to route thus

 As see in first picture id is removed and selected location is inserted in taken location we disabled to again set the location

That will show all the projects 



Second is that admin can define the rubrics admin write the rubrics comma separated and we place it in rubrics table through

 $rubricArray = explode(',', $rubricsInput);

foreach ($rubricArray as $rubric) {

            DB::table('rubrics')->insert([

            'met\_id' => $request->input('project\_id'),

            'name' => $project->title,

            'description' => $rubric,]);

        }

Thus for example if I write red green brwn comma sepearated then on rubricstable it is showing



Having the id and title of the project for which is the rubrics is defined

For this method in detail controller and route is already defined and when admin adds rubrics and set location it redirects back to same page

return redirect('/landing')->with('success', 'Location set successfully.');

**DataBase Report:**

**Entity-Relationship Model:**

These ae the database tables made through command

**Php artisan make:migration create\_tablename(s)**

**admins:**

This table stores information about admin accounts responsible for managing the physical locations of FYP projects on the demonstration floor.

Attributes: id password name email

**evaluations:**

Description: This table tracks the evaluations conducted by evaluators for FYP projects.

Attributes: id evaluator\_id project\_id preferences rating

**evaluators:**

Description: This table stores information about evaluator accounts, including preferences such as preferred project categories and specialty areas.

Attributes: id name email password

**fyp\_groups:**

Description: This table manages details about Final Year Project groups, including assigned keywords.

Attributes: id group\_name email password

**locations:**

Description: Stores information about the physical location of each FYP project on the demonstration floor.

Attributes: id name created\_at updated\_at

**projects:**

Description: This table contains information about Final Year Projects.

Attributes: id fyp\_id title keywords description image\_captions

**project\_images:**

Description: Manages images related to FYP projects.

Attributes: id project\_id image\_path caption

**rubrics:**

Description: Stores information about rubrics defined by the admin for evaluation.

Attributes: met\_id description name id

**rubric\_metrics:**

Description: Contains metrics associated with each rubric.

Attributes: id rubric\_id project\_id rating created\_at updated\_at

**taken\_locations:**

Description: Keeps track of locations already assigned to FYP projects.

Attributes: id (Primary Key), location\_id (Foreign Key).

**Relationships:**

**evaluations → evaluators:**

One-to-Many relationship: Each evaluation is associated with one evaluator, but each evaluator can conduct multiple evaluations.

**evaluations → projects:**

One-to-Many relationship: Each evaluation is linked to a specific FYP project, but each project can have multiple evaluations.

**locations → projects:**

One-to-One relationship: Each location is assigned to a specific FYP project.

**evaluators → fyp\_groups:**

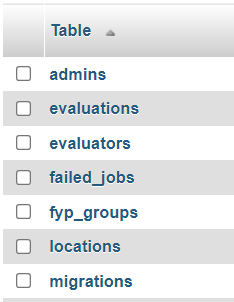
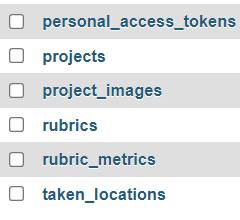
Many-to-Many relationship: Evaluators can have preferences for multiple project categories, and each category can be preferred by multiple evaluators.

**rubrics → rubric\_metrics:**

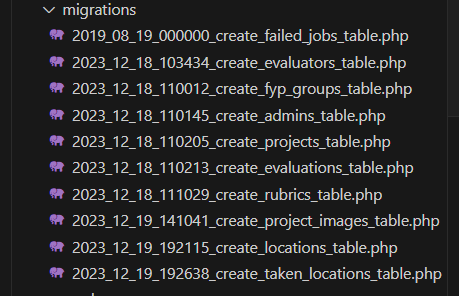
One-to-Many relationship: Each rubric can have multiple associated met

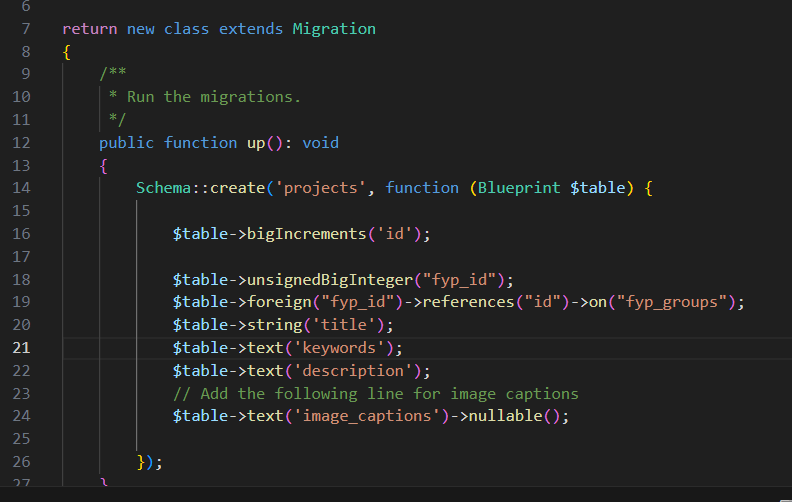
**Bonus Feature - Rubric-Based Evaluation System:**

To implement the rubric-based evaluation system, the database includes tables for rubrics and rubric metrics. Admins can define rubrics, and evaluators can mark scores according to at least three metrics in the rubric during project evaluation.

Then to define tha attributes of each table in a database there should be migration class for it which was made from the above command



which has two methods up and down .Defining attributes in up method   
  


**up Method:**

* Schema::create('admins', function (Blueprint $table) { ... }: Defines the creation of a new table named 'admins.'
* $table->id();: Adds an auto-incrementing primary key column named 'id.'
* $table->string('name');: Adds a string column named 'name' to store the admin's name.
* $table->string('email')->unique();: Adds a string column named 'email' with a unique constraint to store the admin's email.
* $table->string('password');: Adds a string column named 'password' to store the admin's password.

**down Method:**

Schema::dropIfExists('admins');: Defines the rollback action, dropping the 'admins' table if it exists.

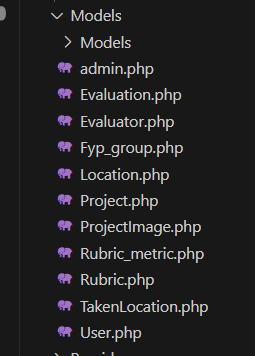
Same all other tables have been given column names

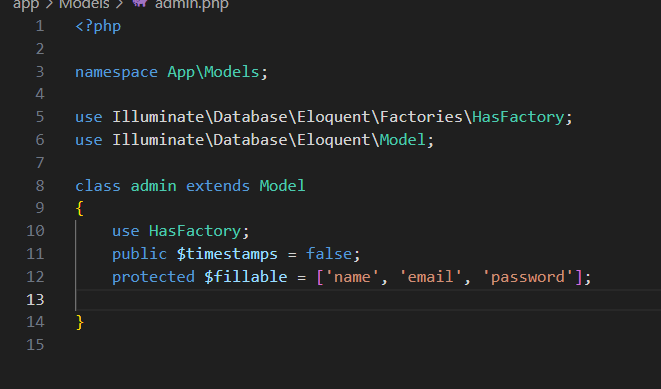
Then to access each table to pally logic and work on database there should be a model class for each table to which we can access and apply query

Command

**Php artisan make:Model ModelName**

(ModelName is the table name in Capitalize and also singular)



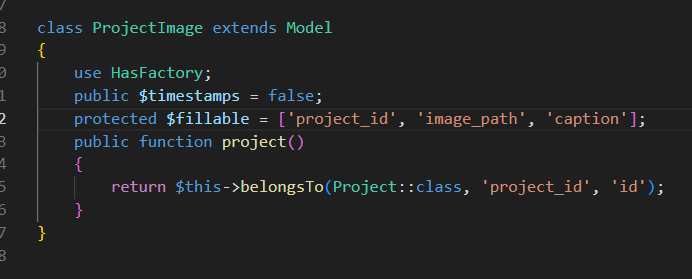


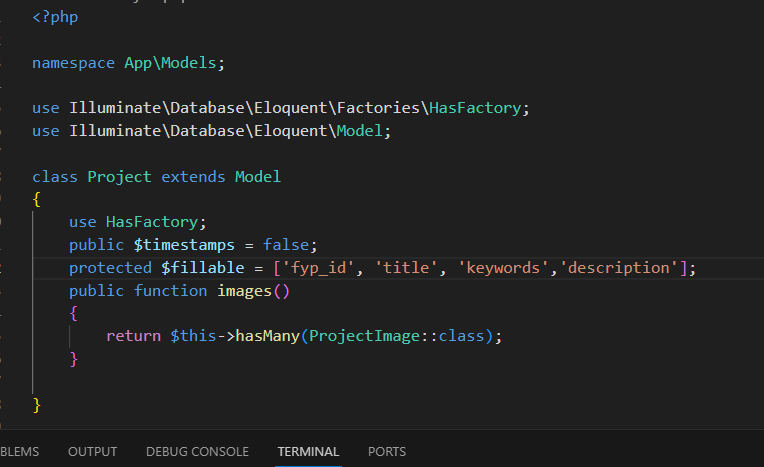
**$fillable Property:**

This is a protected property within an Eloquent model class.

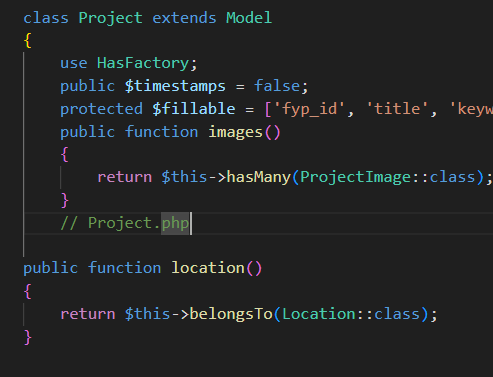
It is an array that lists the names of the attributes that are allowed to be mass-assigned.

**Relation ships :**

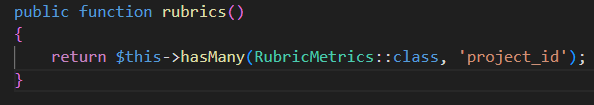




the hasMany relationship in the Project model signifies that a project can have multiple images, while the belongsTo relationship in the ProjectImage model establishes the connection of each image to a specific project. These relationships simplify querying and organizing related data in the database, enhancing the efficiency of data retrieval and manipulation.

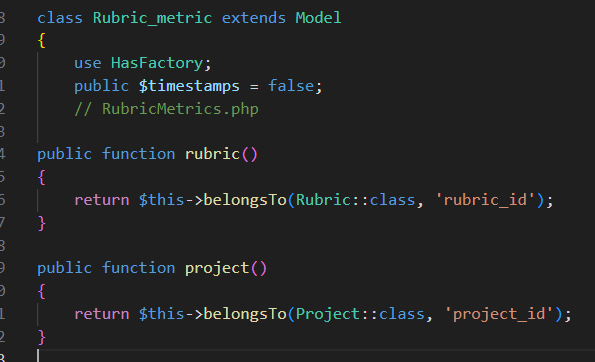


To get the associated location of that project

In roject model 

Since their could be a lot of rubrics associated with project

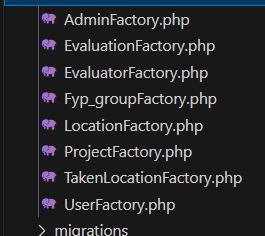
In rubric metric model



**Seeding Data:**

Although all the details and rubrics every thing will be done by authorized users but to test factory method is used to create dummy data through command

**Php artisan make:Factory TableNameFactory**



'name' => $this->faker->name: Generates a fake name using the Faker library, providing realistic-looking names for each generated user.

'email' => $this->faker->unique()->safeEmail: Generates a unique and safe (valid) email address for each user, ensuring email address diversity and avoiding duplicates.

'password' => bcrypt('password'): Sets a default password for each user.

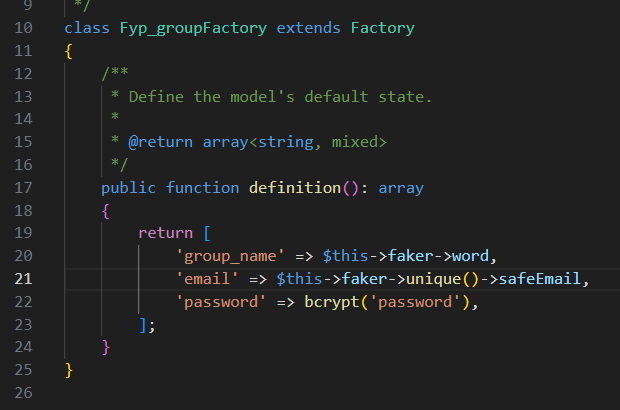
**Faker Library:**

The $this->faker instance is part of the Faker library integrated into Laravel for generating fake data.

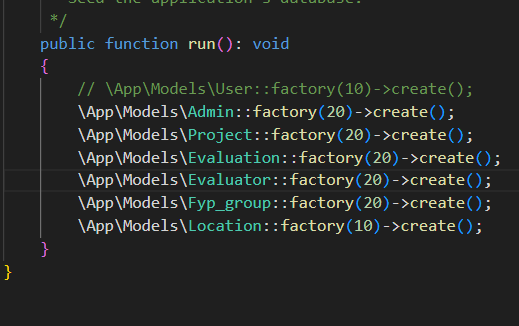
It provides a variety of methods (name, unique, safeEmail, etc.) to create realistic and unique data for testing and development.

**Password Hashing:**

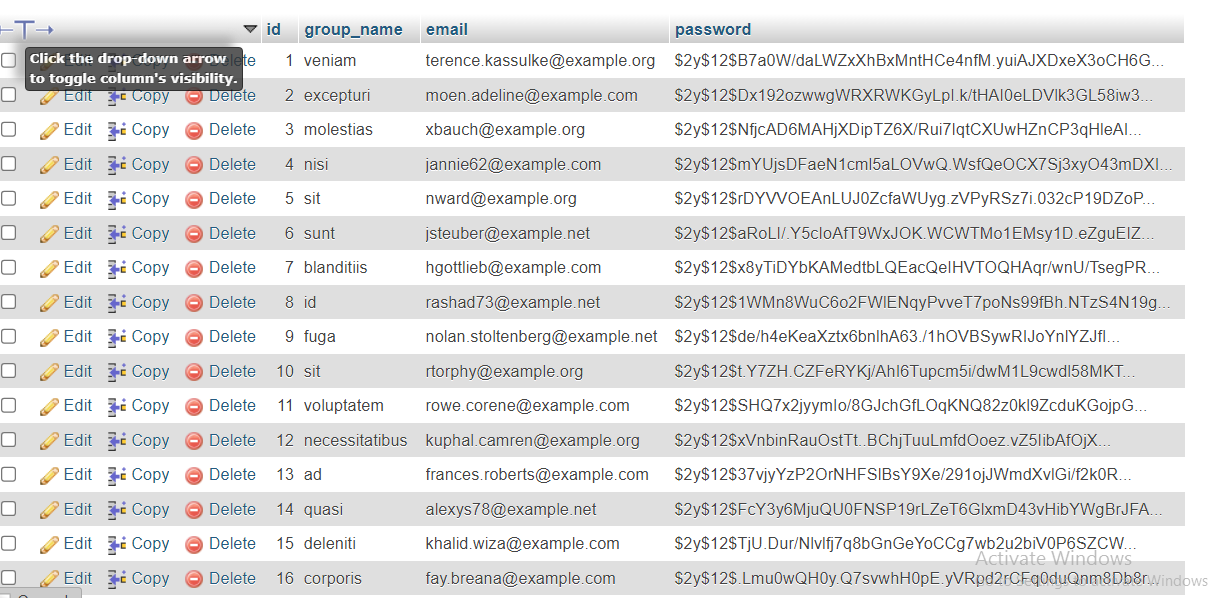
'password' => bcrypt('password'): Hashes the password using Laravel's bcrypt function. This ensures that the stored passwords in the generated data are securely hashed.



In seeders call this function and tell how many dummy instances to ceate



And this is the dummy data



**Conclusion:**

The database design is intended to support the key features outlined for the NUST-SEECS Open House Management Platform. It facilitates the efficient assignment of projects to evaluators, tracks the evaluation process, and includes a bonus feature for rubric-based evaluations. The relationships between tables ensure data integrity and effective data management.

**DataBase Report:**

**Entity-Relationship Model:**

These ae the database tables made through command

**Php artisan make:migration create\_tablename(s)**

**admins:**

This table stores information about admin accounts responsible for managing the physical locations of FYP projects on the demonstration floor.

Attributes: id password name email

**evaluations:**

Description: This table tracks the evaluations conducted by evaluators for FYP projects.

Attributes: id evaluator\_id project\_id preferences rating

**evaluators:**

Description: This table stores information about evaluator accounts, including preferences such as preferred project categories and specialty areas.

Attributes: id name email password

**fyp\_groups:**

Description: This table manages details about Final Year Project groups, including assigned keywords.

Attributes: id group\_name email password

**locations:**

Description: Stores information about the physical location of each FYP project on the demonstration floor.

Attributes: id name created\_at updated\_at

**projects:**

Description: This table contains information about Final Year Projects.

Attributes: id fyp\_id title keywords description image\_captions

**project\_images:**

Description: Manages images related to FYP projects.

Attributes: id project\_id image\_path caption

**rubrics:**

Description: Stores information about rubrics defined by the admin for evaluation.

Attributes: met\_id description name id

**rubric\_metrics:**

Description: Contains metrics associated with each rubric.

Attributes: id rubric\_id project\_id rating created\_at updated\_at

**taken\_locations:**

Description: Keeps track of locations already assigned to FYP projects.

Attributes: id (Primary Key), location\_id (Foreign Key).

**Relationships:**

**evaluations → evaluators:**

One-to-Many relationship: Each evaluation is associated with one evaluator, but each evaluator can conduct multiple evaluations.

**evaluations → projects:**

One-to-Many relationship: Each evaluation is linked to a specific FYP project, but each project can have multiple evaluations.

**locations → projects:**

One-to-One relationship: Each location is assigned to a specific FYP project.

**evaluators → fyp\_groups:**

Many-to-Many relationship: Evaluators can have preferences for multiple project categories, and each category can be preferred by multiple evaluators.

**rubrics → rubric\_metrics:**

One-to-Many relationship: Each rubric can have multiple associated met

**Bonus Feature - Rubric-Based Evaluation System:**

To implement the rubric-based evaluation system, the database includes tables for rubrics and rubric metrics. Admins can define rubrics, and evaluators can mark scores according to at least three metrics in the rubric during project evaluation.

A screenshot of a survey

Description automatically generated A checklist with text on it

Description automatically generated

Then to define tha attributes of each table in a database there should be migration class for it which was made from the above command

A screenshot of a computer

Description automatically generated

which has two methods up and down .Defining attributes in up method   
  
A screen shot of a computer program

Description automatically generated

**up Method:**

* Schema::create('admins', function (Blueprint $table) { ... }: Defines the creation of a new table named 'admins.'
* $table->id();: Adds an auto-incrementing primary key column named 'id.'
* $table->string('name');: Adds a string column named 'name' to store the admin's name.
* $table->string('email')->unique();: Adds a string column named 'email' with a unique constraint to store the admin's email.
* $table->string('password');: Adds a string column named 'password' to store the admin's password.

**down Method:**

Schema::dropIfExists('admins');: Defines the rollback action, dropping the 'admins' table if it exists.

Same all other tables have been given column names

Then to access each table to pally logic and work on database there should be a model class for each table to which we can access and apply query

Command

**Php artisan make:Model ModelName**

(ModelName is the table name in Capitalize and also singular)

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

**$fillable Property:**

This is a protected property within an Eloquent model class.

It is an array that lists the names of the attributes that are allowed to be mass-assigned.

**Relation ships :**

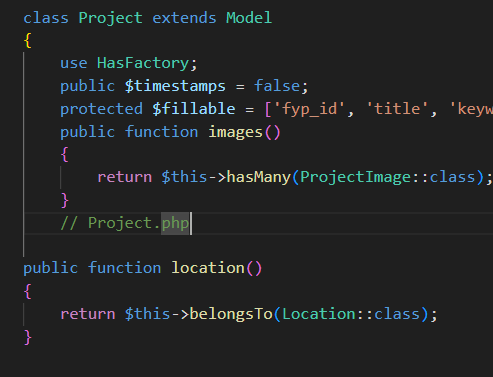
A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

the hasMany relationship in the Project model signifies that a project can have multiple images, while the belongsTo relationship in the ProjectImage model establishes the connection of each image to a specific project. These relationships simplify querying and organizing related data in the database, enhancing the efficiency of data retrieval and manipulation.



To get the associated location of that project

In roject model A black screen with white text

Description automatically generated

Since their could be a lot of rubrics associated with project

In rubric metric model

A computer screen shot of code

Description automatically generated

**Seeding Data:**

Although all the details and rubrics every thing will be done by authorized users but to test factory method is used to create dummy data through command

**Php artisan make:Factory TableNameFactory**

A screen shot of a computer

Description automatically generated

'name' => $this->faker->name: Generates a fake name using the Faker library, providing realistic-looking names for each generated user.

'email' => $this->faker->unique()->safeEmail: Generates a unique and safe (valid) email address for each user, ensuring email address diversity and avoiding duplicates.

'password' => bcrypt('password'): Sets a default password for each user.

**Faker Library:**

The $this->faker instance is part of the Faker library integrated into Laravel for generating fake data.

It provides a variety of methods (name, unique, safeEmail, etc.) to create realistic and unique data for testing and development.

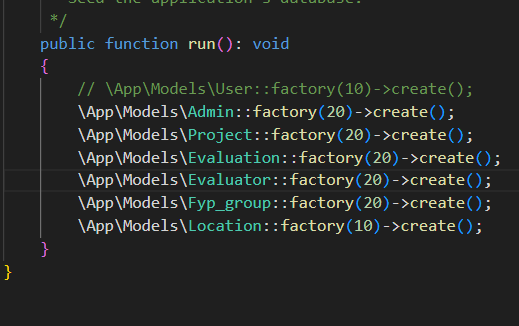
**Password Hashing:**

'password' => bcrypt('password'): Hashes the password using Laravel's bcrypt function. This ensures that the stored passwords in the generated data are securely hashed.

A computer screen with colorful text

Description automatically generated

In seeders call this function and tell how many dummy instances to ceate



And this is the dummy data

A screenshot of a computer

Description automatically generated

**Conclusion:**

The database design is intended to support the key features outlined for the NUST-SEECS Open House Management Platform. It facilitates the efficient assignment of projects to evaluators, tracks the evaluation process, and includes a bonus feature for rubric-based evaluations. The relationships between tables ensure data integrity and effective data management.