Homework

1. What is a Laravel pattern?
   - Laravel pattern is model-view-controller(MVC).
2. What is M, V, C?
   - MVC is the Laravel pattern:
     - M stands for Model
     - V stands for View
     - C stands for Controller
3. What is a query builder?
   - Query Builder provides a graphical user interface for creating SQL queries
4. What is Laravel Eloquent?
   - Eloquent is an object-relational mapper (ORM) that is included by default within the Laravel framework.
5. What is a .env file?
   - The . env file is used in projects to store configuration settings, environment variables, and sensitive information securely.
6. What is migration?
   - Laravel Migration is a set of instructions that define the changes you want to make to your database schema. These changes can include creating new tables, altering existing tables, adding or modifying columns, and seeding the database with initial data.
7. What is Seeder?
   - Seeder is a simple method that provided by Laravel to allow you to seed test data into a database using seeder classes.
8. What is file.blade.php?
   - File.blad.php is the Blade template file using the .blade.php file extension and is typically stored in the resources/views directory.
9. What is Factory?
   - Factory is a class that extends Laravel's base factory class and defines a definition method.
10. What is API?
    - APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols.
11. .What is the end-point, method,request-body, handle error validation and response?

- An Endpoint is a specific URL where an API can be accessed. It typically includes the base URL and the path to a specific resource.
- An HTTP method indicates the action to be performed for a given resource. Common methods include:
    - **GET**: Retrieve data from the server
    - **POST**: Send data to the server to create a resource.
    - **PUT**: Update an existing resource on the server.
    - **DELETE**: Remove a resource from the server.
    - The request body contains the data sent to the server in a POST or PUT request. It is usually formatted in JSON.
    - Error validation ensures that the data sent in a request is correct and that appropriate errors are returned if it is not. This involves checking for missing fields, incorrect data types, or any other rules specified by the API.
    - The response is the data returned by the server after processing the request. It includes a status code and optionally a body.

12. Which method to use when I want to create a new Post?
    - The method to use when you want to create a new post is method “POST”.
13. Why do you need to use Resource and Request and getAttribute?
    - Using resources, requests, and attributes helps in organizing and managing data and interactions efficiently.
14. What is compact()?
    - The compact() function is used to convert a given variable to to array in which the key of the array will be the name of the variable and the value of the array will be the value of the variable.
15. list down at least 10 query builders and Laravel Eloquent.
    - **Query Builder Methods**
        - **select**:

```
DB::table('users')->select('name', 'email')->get();
```

        - **where**:

```
DB::table('users')->where('age', '>', 25)->get();
```

        - **join**:

```
DB::table('users')
    ->join('posts', 'users.id', '=', 'posts.user_id')
    ->get();
```

        - **orderBy**:

```
DB::table('users')->orderBy('name', 'asc')->get();
```

        - **groupBy**:

```
DB::table('orders')
    ->select(DB::raw('count(*) as order_count, status'))
    ->groupBy('status')
    ->get();
```

- **having**:
  ```
  DB::table('orders')
      ->select(DB::raw('count(*) as order_count, status'))
      ->groupBy('status')
      ->having('order_count', '>', 5)
      ->get();
  ```
- **limit**:
  ```
  DB::table('users')->limit(10)->get();
  ```
- **offset**:
  ```
  DB::table('users')->offset(5)->limit(10)->get();
  ```
- **pluck**:
  ```
  DB::table('users')->pluck('name');
  ```
- **count**:
  ```
  DB::table('users')->count();
  ```

- **Laravel Eloquent Methods**
  - **find**:
    ```
    User::find(1);
    ```
  - **all**:
    ```
    User::all();
    ```
  - **create**:
    ```
    User::create(['name' => 'John Doe', 'email' => 'john@example.com']);
    ```
  - **update**:
    ```
    $user = User::find(1);
    $user->update(['email' => 'newemail@example.com']);
    ```
  - **delete**:
    ```
    $user = User::find(1);
    $user->delete();
    ```
  - **first**:
    ```
    User::where('name', 'John Doe')->first();
    ```
  - **get**:
    ```
    User::where('age', '>', 25)->get();
    ```
  - **paginate**:
    ```
    User::paginate(15);
    ```
  - **with**:
    ```
    User::with('posts')->get();
    ```
  - **whereIn**:
    ```
    User::whereIn('id', [1, 2, 3])->get();
    ```

16. Why do we need to use wherehas?

- The whereHas method in Laravel is useful when you need to query models based on the existence of a relationship and apply additional constraints to the related models.

17. Please give an example of Column Types at 10 differences don't forget to the description.
    - **string**
      Stores a variable-length string. Ideal for text fields like names, titles, etc.
    - **integer**
      Stores integer values. Suitable for numeric data that does not require decimals.
    - **boolean**
      Stores a boolean value (true or false).
    - **text**
      Stores large amounts of text. Suitable for long descriptions or content.
    - **date**
      Stores date values without time. Useful for dates like birthdays or anniversaries.
    - **DateTime**
      Stores date and time values. Ideal for timestamps and events that include time.
    - **timestamp**
      Stores UNIX timestamp values. Often used for created_at and updated_at columns.
    - **float**
      Stores floating-point numbers. Suitable for data that requires decimals but not high precision.
    - **JSON**
      Stores JSON data. Ideal for storing structured data that needs to be queried or manipulated as JSON.
    - **binary**
      Stores binary data. Suitable for storing raw binary data like images, files, etc.

18. can you descript about this $table->integer('age')->nullable()->default(0);
    - The line of code $table->integer('age')->nullable()->default(0) is part of a Laravel migration, and it is used to define a column in a database table.
        - **$table**:
            - Represents the table being modified or created. This is typically an instance of Blueprint provided by the Laravel Schema Builder.
        - **integer('age')**:
            - Defines a column named age.
            - The column type is integer, meaning it will store integer values. In SQL, this typically translates to an INT type.
        - **nullable()**:
            - Indicates that the column can contain NULL values.
            - If nullable() is not specified, the column will be NOT NULL by default, meaning it must always contain a value.

        - **default(0)**:
            - Sets the default value of the column to 0.
            - If no value is provided for this column when inserting a new record, 0 will be used as the default value.

19. Can you tell me why when we use file api.php and web.php? explain?

- **api.php**:
    - The api.php file is used to define routes for your application's API endpoints.
    - API routes are typically used for building RESTful APIs that are consumed by other applications or client-side frameworks (e.g., mobile apps, single-page applications).
    - API routes often have different authentication and authorization requirements compared to web application routes.
    - APIs typically return JSON or other structured data formats, rather than rendering full HTML pages.
- **web.php**:
    - The web.php file is used to define routes for your application's web interface.
    - Web routes are typically used for serving HTML pages that are rendered by your Laravel application.
    - Web routes often have different authentication and authorization requirements compared to API routes.
    - Web routes may include pages like /, /about, /contact, /login, /dashboard, etc.
    - Web routes are typically used for building traditional server-rendered web applications.

20. What is relationship Laravel? Please give an example!

- **One-to-One Relationship**:
    - Example:
    - Model: User
    - Relationship: hasOne('App\Models\Profile')
    - Model: Profile
    - Relationship: belongsTo('App\Models\User')
- **One-to-Many Relationship**:
    - Example:
    - Model: User
    - Relationship: hasMany('App\Models\Post')
    - Model: Post
    - Relationship: belongsTo('App\Models\User')
- **One-to-Many Relationship**:
    - Example:

    - Model: User
    - Relationship: hasMany('App\Models\Post')
    - Model: Post
    - Relationship: belongsTo('App\Models\User')

21. can you explain to me why we need to use Route::apiResource in route API and Route::resource in route web.

The differences between Route::apiResource and Route::resource in Laravel are primarily related to the specific use cases and conventions for building APIs vs. web applications.

- **Route::apiResource**:
    - Route::apiResource is specifically designed for defining routes that expose a RESTful API.
    - It automatically generates the standard RESTful routes (index, store, show, update, destroy) for a given resource.
    - These routes are typically used for building APIs that serve JSON or other structured data formats, rather than rendering full HTML pages.
    - Route::apiResource also applies the api middleware group by default, which usually includes middleware like authentication, throttling, and JSON response formatting.
    - The routes generated by Route::apiResource are prefixed with /api by default, following the convention for API endpoints.
    - Example: Route::apiResource('products', 'ProductController');
- **Route::resource**:
    - Route::resource is used for defining routes for a traditional server-rendered web application.
    - It also generates the standard RESTful routes (index, create, store, show, edit, update, destroy) for a given resource.
    - These routes are typically used for rendering HTML pages that can be displayed in a web browser.
    - Route::resource does not apply any specific middleware group by default, allowing you to customize the middleware as needed for your web application.
    - The routes generated by Route::resource are not prefixed with /api by default, as they are intended for the main web application pages.
    - Example: Route::resource('products', 'ProductController');

22. .can you explain this command "php artisan route:list" and "php artisan install:api"

- **php artisan route:list**:
    - This is a Laravel Artisan command that displays a list of all the defined routes in your application.
    - When you run this command, it will show you the HTTP method (e.g., GET, POST, PUT, DELETE), the URI pattern, the name of the route (if it has been named), and the associated controller action.

  - This command is very useful for quickly understanding the available routes in your application, especially when working on a large or complex project.
- **php artisan install:api**:
  - This is not a standard Laravel Artisan command. It's likely a custom command that has been added to your application's codebase.
  - The purpose of this command would depend on how it has been implemented in your application.
  - In general, a custom command like php artisan install:api is used to perform a specific set of tasks related to setting up or configuring the API-related functionality of your Laravel application.

23. Can you explain to me about redirect and route name?

- **Redirection**:
  - Redirection is the process of redirecting the user from one URL to another.
  - In Laravel, you can use the redirect() helper function to perform redirections.
  - The redirect() function can be used in various ways, such as:
    1. redirect('/home'): Redirects the user to the /home URL.
    2. redirect()->route('home'): Redirects the user to the route named home.
    3. redirect()->back(): Redirects the user to the previous URL.
  - Redirections are often used when processing form submissions, after a successful action, or when the user needs to be redirected to a different page.
- **Route Names**:
  - Route names are unique identifiers assigned to your application's routes.
  - Route names allow you to refer to a specific route by its name instead of its URL pattern.
  - Assigning names to your routes is a best practice, as it makes your code more maintainable and less prone to breaking changes.
  - You can assign a name to a route using the ->name() method when defining the route:

24. Clean the code using Route Prefixes:

```
Route::prefix('product')->name('product.')->group(function () {
   Route::get('list', [ProductController::class, 'index'])->name('list');
   Route::post('create', [ProductController::class, 'store'])->name('create');
   Route::get('show/{id}', [ProductController::class, 'show'])->name('show');
   Route::put('update/{id}', [ProductController::class, 'update'])->name('update');
   Route::delete('delete/{id}', [ProductController::class, 'destroy'])->name('destro
y');
});
```

25. what is the difference between "php artisan migrate" and "php artisan migrate:fresh"?
    - **php artisan migrate**:
        - This command runs the database migrations that have not yet been applied to the database.
        - It will apply any new migrations that have been added to the database/migrations directory, without affecting the existing database structure.
        - This is the command you would typically use for deploying new migrations to an existing database.
    - **php artisan migrate:fresh**:
        - This command will first drop all tables in the database, and then run the migrations to create the tables again.
        - It effectively resets the database to a clean state, as if you had just created a new database.
        - This is useful when you want to start fresh with your database, for example, during development or when making major changes to your database structure.
        - It is important to note that using migrate:fresh will result in **data loss**, as it will drop all existing tables in the database.
26. why do we need to use this?

```
protected $fillable = [
          'name',
           'email',
             'password',
   ];
```

The $fillable property in Laravel's Eloquent models is used to specify which model attributes are "mass assignable". This means that these attributes can be assigned values during the model creation or update process without the risk of "mass assignment vulnerabilities".

27. Why do we need to use this?

```
  protected $hidden = [
      'password',
       'remember_token',
   ];
```

The $hidden property in Laravel's Eloquent models is used to specify which model attributes should be hidden when the model is serialized, such as when converting the model to an array or JSON.

28. Why we need to use

```
   protected $casts = [
        'email_verified_at' => 'datetime',
    ];
```

The $casts property in Laravel's Eloquent models is used to specify how certain attributes should be cast when they are accessed or set on the model.

29. Why we need to use SoftDeletes?

The SoftDeletes trait in Laravel's Eloquent is used to implement "soft deletes" for your Eloquent models. Soft deletes are a way of marking a model as deleted, without actually removing it from the database.

30. What is file composer.json?

The composer.json file is a crucial part of any PHP project that uses Composer, the de facto standard for managing dependencies in the PHP ecosystem. It is a JSON-formatted file that serves as the project's configuration and dependency management manifest.

31. What is README.md?

- The README.md file is a crucial part of any software project, as it provides important information about the project to both users and other developers.
- The README.md file is typically written in Markdown, a lightweight markup language that is easy to read and write. It is the first thing that people see when they visit a project's repository, and it can greatly influence their decision to use or contribute to the project.