LARAVEL BASICS

1. Routing:

Laravel uses a routes/web.php file to define web routes and routes/api.php for API routes. Routes define how your application responds to HTTP requests.

```
Route::get('/', function () {
    return view('welcome');
});
```

2. Controller:

Controllers handle the logic of your application. You can create a controller using Artisan (Laravel's command-line tool):

```
php artisan make:controller MyController
```

This will generate a new controller in the app/Http/Controllers directory.

3.Views:

Views are stored in the resources/views directory. They are responsible for presenting the data to the user. Blade is the templating engine used by Laravel.

4.Models:

In Laravel, models are classes that represent and interact with the data in your application's database. They allow you to query and manipulate database records in an object-oriented manner. Here are the key concepts related to models in Laravel:

Creating Models:

You can create a new model using the Artisan command:

```
php artisan make:model ModelName
```

This will generate a new model class in the app directory.

Defining a Table:

By default, Eloquent assumes that the table associated with a model is named in a plural, snake_case version of the model's name. You can define a specific table name in your model:

```
class UserModel extends Model
{
    protected $table = 'user_table';
}
```

5. Database Migration and Seeds:

Laravel's migration system allows you to version-control your database schema. Artisan commands like make:migration and migrate help in managing database changes.

```
php artisan make:migration create_table_name
php artisan migrate
```

Seeder classes help you populate your database with sample data:

```
php artisan make:seeder MySeeder
php artisan db:seed --class=MySeeder
```

6.Middleware:

Middleware in Laravel provides a way to filter HTTP requests entering your application. They act as a bridge between the incoming request and your application's logic. Laravel middleware can perform tasks such as authentication, logging, CORS handling, CSRF protection, and more.

Creating Middleware:

You can create a new middleware using the make:middleware Artisan command:

```
php artisan make:middleware MyMiddleware
```

Middleware Structure:

A middleware class typically contains a handle method that receives the incoming request and decides whether to pass it further into the application or terminate it.

```
public function handle($request, Closure $next)
{
// Perform actions before the request is handled
    $response = $next($request);
    // Perform actions after the request is handled
    return $response;
}
```

Registering Middleware:

You can register middleware in the $middleware property of the app/Http/Kernel.php file. Global middleware runs on every HTTP request, while route middleware is applied to specific routes.

```
protected $middleware = [
    // Global middleware
    \App\Http\Middleware\MyMiddleware::class,
];

protected $routeMiddleware = [
    // Route-specific middleware
    'auth' => \App\Http\Middleware\Authenticate::class,
];
```

Middleware Groups:

Middleware groups allow you to group several middleware under a single key, making it easier to apply multiple middleware to routes.

```
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,

\Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::
class,
        // ... other middleware
    ],
];
```

7.Authentication and Authorization:

Laravel provides a simple and easy-to-use authentication system. You can use Artisan commands to scaffold basic login and registration functionality.

```
php artisan make:auth
```

Laravel also provides a robust authorization system using Gates and Policies.

8.Pagination:

Use the paginate method on Eloquent models to split query results into pages.

```
$users = User::paginate(10); // Paginate users, 10 per page
```

Displaying Pagination Links:

Show pagination links in Blade templates with the links method.

```
{{ $users->links() }}
```

9.CSRF:

CSRF (Cross-Site Request Forgery) is a security feature implemented in Laravel to protect against malicious actions performed on behalf of authenticated users without their consent. Laravel automatically generates CSRF tokens for each active user session to validate requests.

CSRF Token Generation:

Laravel includes a CSRF token in every form generated by the framework. This token is automatically included as a hidden field named _token. It can be added to your forms using the @csrf Blade directive:

```
<form method="POST" action="/example">
    @csrf
    <!-- Other form fields go here -->
</form>
```

10.Request Class:

In Laravel, the Illuminate\Http\Request class is a fundamental part of handling HTTP requests. It provides a convenient and consistent way to interact with incoming requests, whether they are GET, POST, PUT, DELETE, or any other HTTP method.

11. Mutators & Accessors:

In Laravel's Eloquent ORM, mutators and accessors are methods that allow you to manipulate attribute values when you set or retrieve them from a model. These methods can be defined in your Eloquent model to customize the behavior of certain attributes.

Mutators:

1.Defining Mutators:

Mutators are used to modify attribute values before they are saved to the database. To define a mutator, create a method in your Eloquent model with the attribute name in camel case, preceded by "set," and suffixed with "Attribute."

```
public function setNameAttribute($value)
{
    $this->attributes['name'] = strtoupper($value);
}
```

In this example, the `setNameAttribute` mutator will convert the value of the "name" attribute to uppercase before saving it to the database.

2. Using Mutators:

Mutators are automatically called when you set the value of an attribute on the model.

```
$model = new MyModel();
$model->name = 'John Doe'; // The mutator is automatically called
$model->save();
```

Accessors:

1. Defining Accessors:

Accessors are used to manipulate attribute values when you access them. To define an accessor, create a method in your Eloquent model with the attribute name in camel case, preceded by "get," and suffixed with "Attribute."

```
public function getNameAttribute($value)
{
    return strtoupper($value);
}
```

In this example, the `getNameAttribute` accessor will return the "name" attribute in uppercase when you access it.

2. **Using Accessors:**

Accessors are automatically called when you access the value of an attribute on the model.

```
$model = MyModel::find(1);
echo $model->name; // The accessor is automatically called
```

In this case, even though the actual value stored in the database is in its original form, the accessor modifies it before it is returned.

12.Logging:

Logging is a crucial aspect of application development for tracking and recording various events, errors, and activities. In Laravel, you can use the built-in logging features to log information, warnings, errors, and more. Laravel uses the Monolog library under the hood, providing a flexible and powerful logging system.

Here's a basic overview of logging in Laravel:

1. Logging Levels:

Laravel supports various logging levels, including `emergency`, `alert`, `critical`, `error`, `warning`, `notice`, `info`, and `debug`. You can use these levels to categorize the severity of log messages.

2. Configuring Logging:

The logging configuration file is located at `config/logging.php`. Here, you can configure the logging channels, handlers, and their respective settings. By default, Laravel comes with channels for `stack`, `single`, and `daily`.

3. Logging Messages:

You can log messages using the `Log` facade or the `logger` helper function:

```
use Illuminate\Support\Facades\Log;
Log::info('This is an informational message.');
```

You can specify the log level explicitly:

```
Log::error('This is an error message.');
```

13.Localization:

Localization in Laravel refers to adapting your application to different languages and regions. Laravel provides robust support for localization through its built-in features, making it easy to translate your application into multiple languages.

Here's a concise overview of Laravel's localization:

1. Configuration:

Localization settings are configured in the `config/app.php` file. Set the `locale` to the default language.

2. Language Files:

Translation strings are stored in language files located in the `resources/lang` directory. Each language has its own subdirectory, and files are organized by file name and language.

3. Translating Strings:

Use the `__` helper function or the `trans` method to translate strings.

```
// Using the __ helper function
  $welcome = __('messages.welcome');

  // Using the trans method
  $goodbye = trans('messages.goodbye');
```

14. Dependency Injection:

Dependency injection (DI) is a design pattern used in Laravel (and many other modern frameworks) that allows you to inject dependencies into a class rather than creating or managing them within the class itself. This promotes a more modular and flexible codebase, making it easier to test, maintain, and swap out components.

In Laravel, dependency injection is commonly used in controllers, services, and other classes. Here's a brief overview:

1. Constructor Injection:

The most common form of dependency injection is through the constructor. Dependencies are passed to the class through its constructor.

```
class MyController
  {
      protected $myService;

      public function __construct(MyService $myService)
      {
          $this->myService = $myService;
      }

      public function someMethod()
      {
          $result = $this->myService->doSomething();
      }
  }
```

2. Method Injection:

Dependencies can also be injected into specific methods when needed.

```
class MyController
{
    public function someMethod(MyService $myService)
    {
        $result = $myService->doSomething();
    }
}
```