

What is .env file/set .env file

What is configuration file

Caching of configuration file

Database connectivity

What is migration

Commands of migration

Create migration

Maintenance Mode

## What is .env File

In Laravel, the .env file is a configuration file used to store sensitive information and environmental variables for your application. It stands for "environment" file. This file is located in the root directory of your Laravel project.

The .env file uses a key-value pair format, where each line represents a configuration variable. It typically contains sensitive information such as database credentials, API keys, and other environment-specific settings. These values can be accessed throughout your Laravel application using the env helper function or by using the `$_ENV` superglobal.

Here's an example of what a .env file in Laravel might look like:

```
APP_NAME=Laravel
APP_ENV=production
APP_KEY=your-application-key
APP_DEBUG=false

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=your-database-name
DB_USERNAME=your-database-username
DB_PASSWORD=your-database-password
```

# What are configuration files in Laravel

In Laravel, configuration files are files that contain various settings and options for your application. These files allow you to customize and define the behavior of different components and services within Laravel.

Configuration files in Laravel are typically stored in the config directory of your Laravel project. They are written in PHP and follow a simple array-based structure.

Here are some examples of configuration files commonly found in Laravel:

1) app.php: This file contains general application settings such as the application name, environment, timezone, and service providers.

2) database.php: This file contains database connection configurations, including the default database connection, MySQL, PostgreSQL, SQLite, and more.

3) cache.php: This file contains configuration options for caching, such as the default cache driver, cache stores, and cache lifetimes.

4) mail.php: This file contains configurations related to sending emails, including the default mail driver, SMTP settings, and email encryption.

5) filesystems.php: This file contains configuration options for file storage and handling, including the default file storage driver, local disks, cloud storage providers, and their respective settings.

6) logging.php: This file contains configuration options for logging, including the log channels, their respective drivers, log levels, and log storage locations.

7) session.php: This file contains configuration options for session management, including the session driver, session lifetime, and session encryption.

These are just a few examples, and Laravel provides many more configuration files for different components and services. You can also create your own custom configuration files to define settings specific to your application.

To access the values defined in configuration files within your application, you can use the config helper function or the Config facade. For example, `config('app.name')` will retrieve the value of the name key from the app.php configuration file.

By utilizing configuration files, Laravel allows you to easily manage and customize various aspects of your application's behavior without modifying the underlying code, promoting flexibility and maintainability.

## Caching of Configuration

In Laravel, you have the option to cache your application's configuration files, which can improve the performance of your application by reducing the overhead of reading and parsing configuration files on each request.

When you cache the configuration, Laravel will merge all the configuration files into a single file, which is then stored in the bootstrap/cache directory. This cached configuration file is loaded much faster than parsing multiple configuration files, resulting in quicker application startup times.

To cache your configuration, you can use the config:cache Artisan command. Open your command-line interface, navigate to the root directory of your Laravel project, and run the following command:

```
php artisan config:cache
```

This command will generate the cached configuration file bootstrap/cache/config.php. You should run this command whenever you make changes to your configuration files or deploy your application to ensure that the cached configuration is up to date.

It's important to note that if you make changes to your configuration files after caching, you'll need to clear the configuration cache by running the config:clear Artisan command:

```
php artisan config:clear
```

This command will remove the cached configuration file, and your application will revert to reading and parsing the individual configuration files on each request.

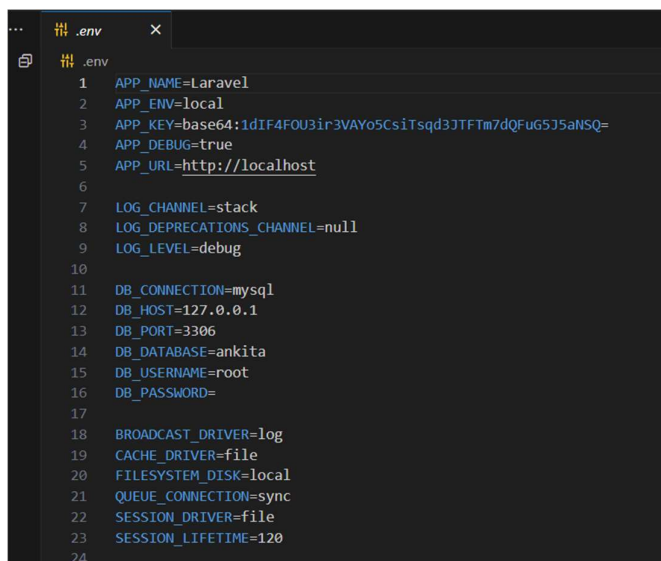
By caching your configuration, you can optimize the performance of your Laravel application by reducing the overhead of loading and parsing configuration files, especially in production environments where the configuration is less likely to change frequently.

## Learn how to configure database connections in Laravel:-

Open the .env file in the root directory of your Laravel project. This file contains environment-specific variables, including database connection details.

Locate the following variables in the .env file:

- 1) DB\_CONNECTION: This variable specifies the database connection driver. The default driver is usually set to mysql.
- 2) DB\_HOST: This variable specifies the database host.
- 3) DB\_PORT: This variable specifies the port number on which the database server is running.
- 4) DB\_DATABASE: This variable specifies the name of the database you want to connect to.
- 5) DB\_USERNAME: This variable specifies the username for the database connection.
- 6) DB\_PASSWORD: This variable specifies the password for the database connection.
- 7) Update the values of these variables according to your database configuration. For example:



```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:1dIF4FOU3ir3VAYo5CsiTsqd3JTFTm7dQFuG5J5aNSQ=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATIONS_CHANNEL=null
9 LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=ankita
15 DB_USERNAME=root
16 DB_PASSWORD=
17
18 BROADCAST_DRIVER=log
19 CACHE_DRIVER=file
20 FILESYSTEM_DISK=local
21 QUEUE_CONNECTION=sync
22 SESSION_DRIVER=file
23 SESSION_LIFETIME=120
24
```

Save the changes to the .env file.

Laravel uses these values to establish a database connection when interacting with the database. By default, Laravel supports various database systems, including MySQL, PostgreSQL, SQLite, and SQL Server. The DB\_CONNECTION variable determines which database driver Laravel should use.

Once you have configured the database connection in the .env file, Laravel will automatically use these settings when performing database operations through its database query builder or Eloquent ORM.

It's important to note that if you make changes to the .env file, you should clear the application cache using the following Artisan command to reflect the updated configuration:

```
php artisan config:cache
```

This command will clear the cached configuration, ensuring that Laravel picks up the latest values from the .env file.

By configuring the database connection in Laravel, you can establish a connection to your desired database system and leverage the database features within your application.

## Understand migrations to manage database schema changes.

Migrations in Laravel are a powerful feature that allows you to manage database schema changes in a structured and version-controlled manner. Migrations enable you to modify the database structure, create or alter tables, add or remove columns, and perform other database schema operations.

Here's an overview of how migrations work in Laravel:

### Migration Files:

Migrations in Laravel are defined as individual files located in the database/migrations directory of your Laravel project.

Each migration file contains two methods: up and down. The up method defines the actions to be performed when applying the migration, while the down method defines the actions to be performed when rolling back the migration.

## Laravel Migration Commands

To view the migration commands, open the terminal, and enter the command

```
"php artisan list"
```

This command lists all the commands available in Laravel

There are six commands of migrate in Laravel:

- migrate:fresh
- migrate:install
- migrate:refresh
- migrate:reset
- migrate:rollback

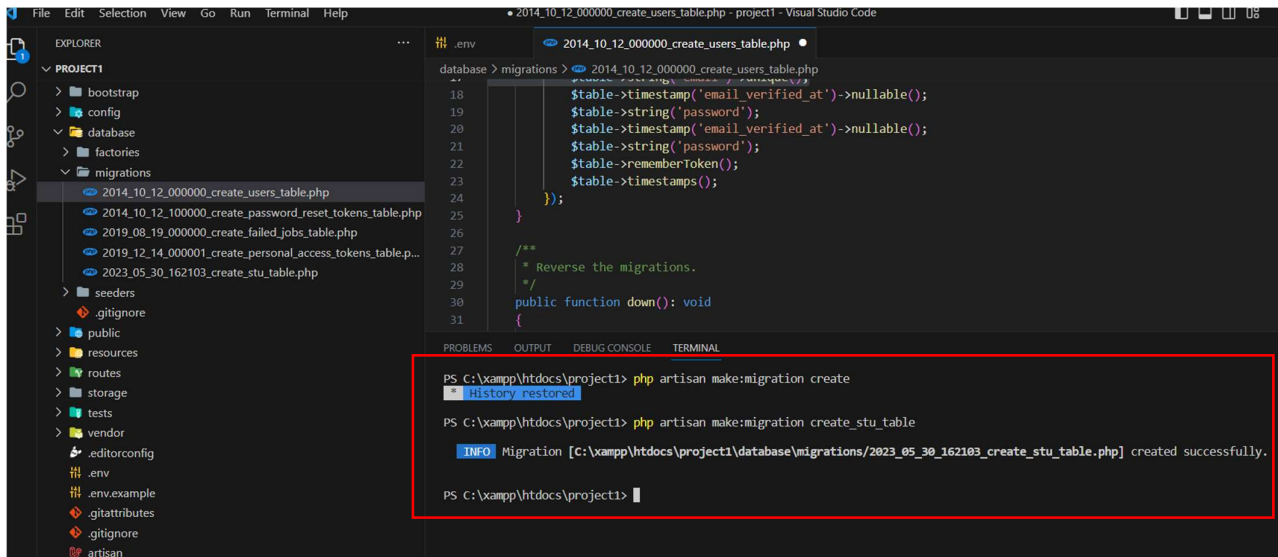
- migrate:status

In Laravel, there are several Artisan commands available to work with migrations. These commands allow you to create migration files, apply migrations, rollback migrations, view migration status, and more. Here are some commonly used Laravel migration commands:

## 1) Create Migration:

**php artisan make:migration create\_table\_name**

This command creates a new migration file in the database/migrations directory with a timestamp prefix and a specified name. The --create option indicates that the migration will create a new table.



The screenshot shows the Visual Studio Code interface with a project named 'PROJECT1'. The Explorer panel on the left shows the file structure, including the 'migrations' directory. The main editor displays the content of a migration file named '2014\_10\_12\_000000\_create\_users\_table.php'. The code includes a 'up()' method that creates a table with columns for 'email\_verified\_at', 'password', and 'remember\_token', and a 'down()' method that reverses the migration. The Terminal panel at the bottom shows the execution of the command 'php artisan make:migration create\_stu\_table', which was successful.

```
database > migrations > 2014_10_12_000000_create_users_table.php
18
19
20
21
22
23
24
25
26
27
28
29
30
31

$table->timestamp('email_verified_at')->nullable();
$table->string('password');
$table->timestamp('email_verified_at')->nullable();
$table->string('password');
$table->rememberToken();
$table->timestamps();
});

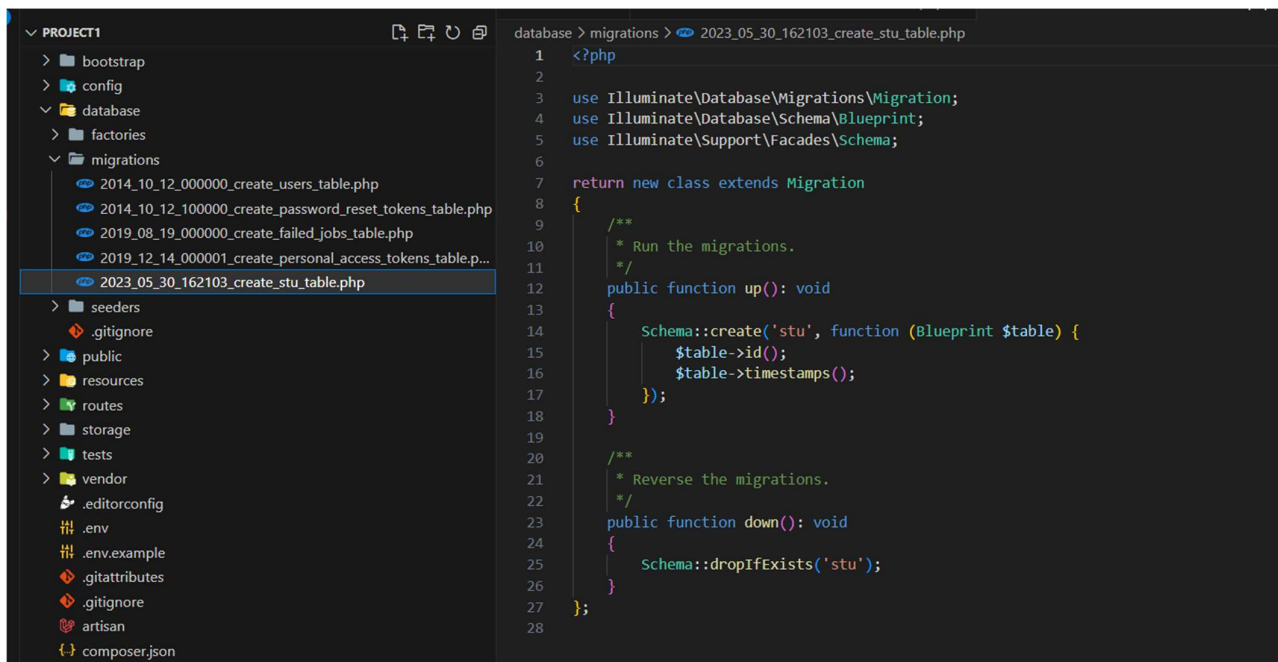
/**
 * Reverse the migrations.
 */
public function down(): void
{
}
```

```
PS C:\xampp\htdocs\project1> php artisan make:migration create
History restored

PS C:\xampp\htdocs\project1> php artisan make:migration create_stu_table

INFO Migration [C:\xampp\htdocs\project1\database\migrations\2023_05_30_162103_create_stu_table.php] created successfully.

PS C:\xampp\htdocs\project1>
```



The screenshot shows the Visual Studio Code interface with the same project. The Explorer panel shows the 'migrations' directory. The main editor displays the content of a migration file named '2023\_05\_30\_162103\_create\_stu\_table.php'. The code includes a 'up()' method that creates a table named 'stu' with an 'id' column and a 'timestamps()' method, and a 'down()' method that drops the table if it exists.

```
database > migrations > 2023_05_30_162103_create_stu_table.php
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('stu', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('stu');
    }
};
```

And adding some data do some change in create table file: -

```
database > migrations > 2023_05_30_162103_create_stu_table.php
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       */
12     public function up(): void
13     {
14         Schema::create('stu', function (Blueprint $table) {
15             $table->id();
16             $table->string('name');
17             $table->string('address');
18             $table->timestamps();
19         });
20     }
21
22     /**
23      * Reverse the migrations.
24      */
25     public function down(): void
26     {
27         Schema::dropIfExists('stu');
28     }
29 };
30
```

Then save it and run the command “php artisan migrate” for saving successful in database

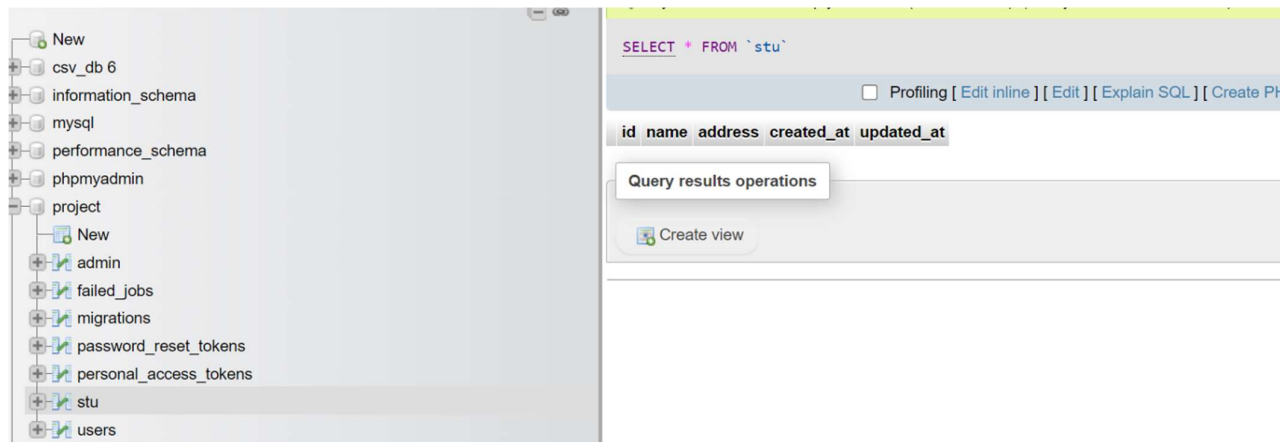
```
PS C:\xampp\htdocs\project1> php artisan migrate

INFO Running migrations.

2023_05_30_162103_create_stu_table ..... 18ms DONE

PS C:\xampp\htdocs\project1>
```

And then for confirmation you also check in MySQL as link manner: -



The screenshot shows the MySQL Workbench interface. On the left, the 'project' database is selected, and the 'stu' table is highlighted in the table list. On the right, the 'Query' tab is active, showing a query: `SELECT * FROM `stu``. Below the query, the table structure is displayed with columns: `id`, `name`, `address`, `created_at`, and `updated_at`. The 'Query results operations' section is visible, showing a 'Create view' button.



## 2) Create Migration (for Existing Table):

**php artisan make:migration add\_column\_to\_table**

**where--table=table\_name**

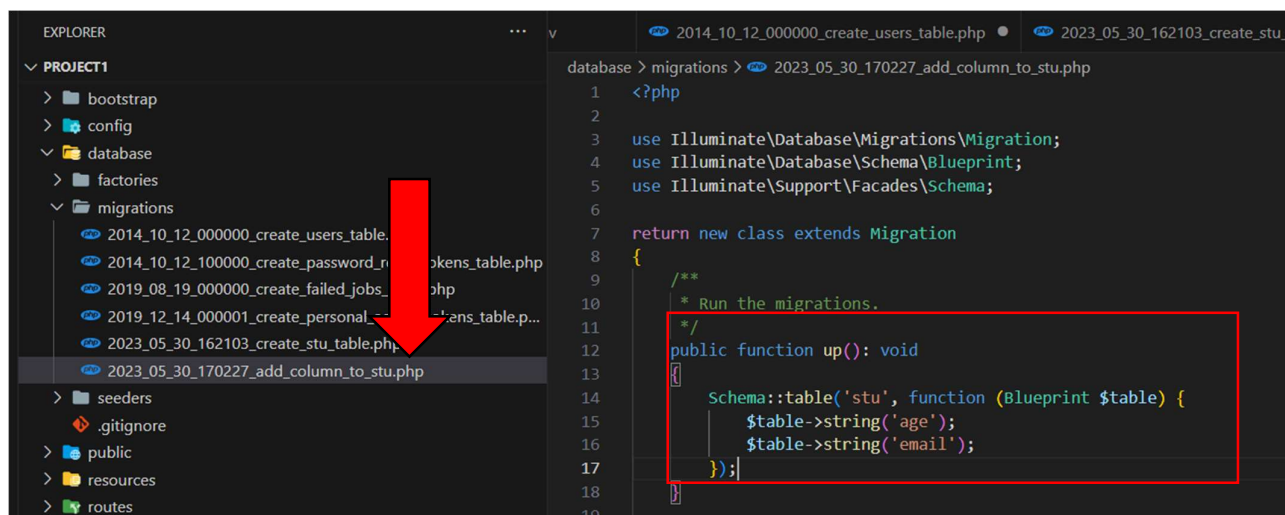
This command generates a new migration file that modifies an existing table. The --table option specifies the name of the table to be modified.

```
PS C:\xampp\htdocs\project1> php artisan make:migration add_column_to_stu

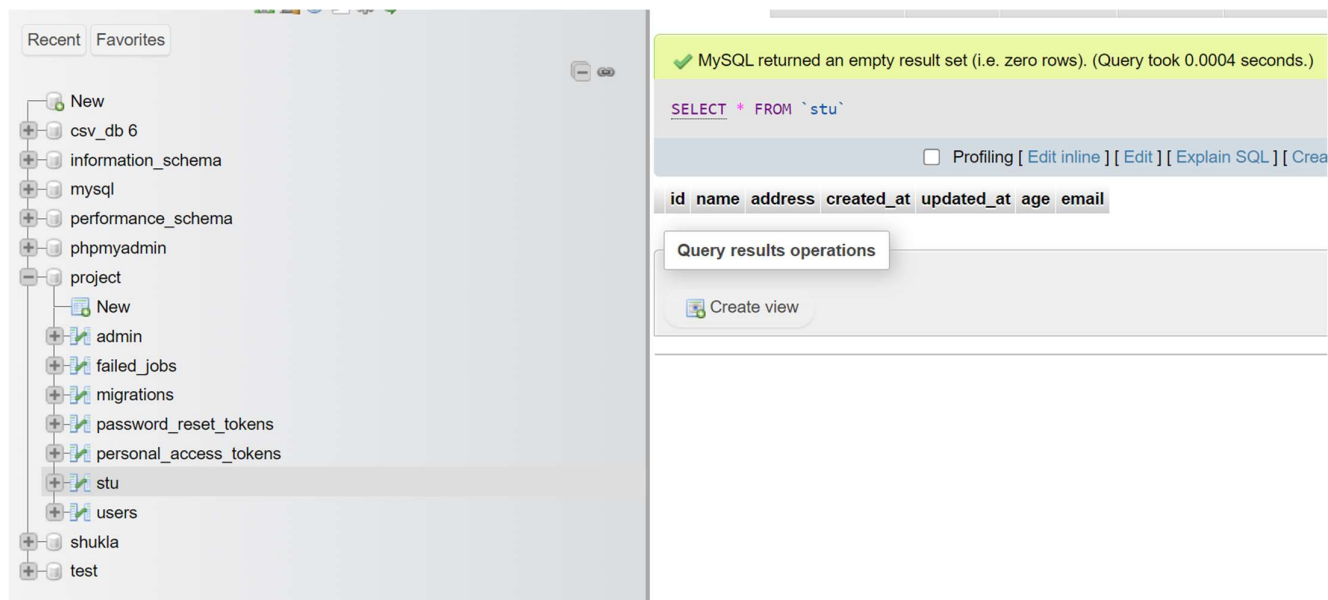
[INFO] Migration [C:\xampp\htdocs\project1\database\Migrations\2023_05_30_170227_add_column_to_stu.php] created successfully

PS C:\xampp\htdocs\project1>
```

You can see in below code the migration file is created and now you can do some change



Then save it and run the command “php artisan migrate” for saving successful in database



## 3) Run Migrations:



## **php artisan migrate**

This command applies any pending migrations and updates the database schema to the latest version.

## **4) Rollback Migrations:**

### **php artisan migrate:rollback**

This command reverts the last batch of migrations, undoing the changes made to the database schema.

## **5) Reset Migrations:**

### **php artisan migrate:reset**

This command rolls back all migrations, resetting the database schema to its initial state.

## **6) Refresh Migrations:**

### **php artisan migrate:refresh**

This command rolls back all migrations and then re-runs them, effectively resetting and reapplying the migrations.

## **7) Rollback and Redo a Single Migration:**

### **php artisan migrate:rollback --step=1**

### **php artisan migrate**

## **8) View Migration Status: -**

### **php artisan migrate:status**

This command displays the status of each migration, indicating whether it has been run or not

## **Maintenance Mode**

Maintenance mode in Laravel allows you to display a maintenance page or perform maintenance tasks on your application while temporarily taking it offline. It's useful when you need to perform updates, database migrations, or any other tasks that require temporary interruption of the normal application flow.

To enable maintenance mode in Laravel, you can use the down Artisan command. Open your command-line interface, navigate to the root directory of your Laravel project, and run the following command:

```
php artisan down
```

When you run this command, Laravel will create a file called down in the storage directory, indicating that the application is in maintenance mode. By default, Laravel will display a default maintenance page with a 503 HTTP status code

Once you have completed the necessary maintenance tasks, you can bring your application back online by using the up Artisan command:

```
php artisan up
```

Running this command will remove the down file from the storage directory, taking your application out of maintenance mode and allowing normal access to the application.

Maintenance mode is a useful feature in Laravel that helps you gracefully handle application downtime or interruptions, ensuring a smooth user experience during maintenance periods

## Eloquent ORM

Eloquent is Laravel's object-relational mapping (ORM) system, which provides an intuitive and expressive way to interact with databases. It allows you to work with database tables as objects and provides methods and relationships to perform various database operations.

Here are some key features and concepts of Eloquent ORM:

### **1) Model and Database Table Relationship:**

In Eloquent, each database table is associated with a corresponding model class. The model class represents a record in the table and provides methods to interact with the data. By convention, the model class is named singular and follows CamelCase naming (e.g., User model for the users table).

### **2) Retrieving Records:**

Eloquent provides a fluent query builder interface to retrieve records from the database. You can use methods like `get()`, `first()`, `find()`, or `where()` to fetch records based on specific conditions. Eloquent also supports eager loading to efficiently retrieve related records.

### **3) Creating and Updating Records:**

Eloquent makes it easy to create and update records. You can create a new record by instantiating a model object and assigning values to its properties, then calling the `save()` method. To update an existing record, retrieve it from the database, modify its properties, and again call the `save()` method.

### **4) Deleting Records:**

Eloquent provides a `delete()` method to remove records from the database. You can call this method on a model instance to delete the corresponding record.

### **5) Relationships:**

Eloquent allows you to define relationships between models, making it easier to work with related data. There are several types of relationships available, including `belongsTo()`, `hasMany()`, `hasOne()`, `belongsToMany()`, and more. These methods define the relationships and provide convenient methods to retrieve related records.

### **6) Query Scopes:**

Eloquent supports query scopes, which are methods defined on a model to encapsulate reusable query conditions. Scopes allow you to define common query constraints and apply them to queries effortlessly.

### **7) Accessors and Mutators:**

Eloquent provides accessors and mutators to define custom getters and setters for model attributes. This allows you to perform transformations or manipulate data before retrieving or storing it in the database.

### **8) Mass Assignment Protection:**

Eloquent offers built-in protection against mass assignment vulnerabilities. You can specify the fillable or guarded attributes on a model to control which attributes can be mass assigned when creating or updating records.

### **9) Events:**

Eloquent triggers events during various stages of a model's lifecycle, such as creating, updating, deleting, or retrieving records. You can define event listeners to perform additional actions or execute custom logic when these events occur.

## **What is model :-**

In Laravel, a "model" refers to a class that represents a specific table in a database. Models in Laravel are used to interact with the database, retrieve data, perform data manipulation, and define relationships between different tables.

Models provide an object-oriented way of working with the database, allowing you to perform database operations using methods and properties defined within the model class. By using models, you can abstract away the direct interaction with the database and work with data in a more intuitive and expressive manner.

## **Here are some reasons why we use models in Laravel:**

- 1) **Data Abstraction:** Models provide a layer of abstraction between your application code and the database. They encapsulate the logic for retrieving and manipulating data, allowing you to work with data as objects rather than writing raw SQL queries.
- 2) **Object-Relational Mapping (ORM):** Laravel's Eloquent ORM is an implementation of the Active Record pattern, where each database table has a corresponding model class. Models enable you to define relationships between tables using methods and properties, making it easy to work with related data.
- 3) **Query Building:** Models provide a fluent query builder interface, allowing you to build complex database queries using chainable methods. This makes it easier to construct queries with conditions, joins, sorting, and pagination.
- 4) **Validation and Mass Assignment:** Models in Laravel include features for data validation and mass assignment protection. You can define rules for validating input data before saving it to the database, ensuring data integrity. Mass assignment protection helps prevent unexpected data changes by specifying which attributes can be mass assigned.
- 5) **Event Hooks:** Laravel models allow you to define event hooks that are triggered during various stages of a model's lifecycle, such as creating, updating, and deleting records. These hooks enable you to perform additional actions or execute custom logic when specific events occur.

By utilizing models in Laravel, you can streamline database operations, improve code organization, enhance data integrity, and simplify working with related data in your application.

## **How to Create model in Laravel:-**

To create a model in Laravel, you can use the Artisan command-line tool provided by Laravel. Here's the step-by-step process:

Open your terminal or command prompt and navigate to the root directory of your Laravel project.

Run the following command to create a model:

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

- If you would like to generate a database migration when you generate the model, you may use

the --migration or -m option:

```
php artisan make:model User --migration
```

Laravel will generate a new file under the app directory with the specified model name. By default, the file will be created in the app namespace.