CAUD Laraval

https://kinsta.com/blog/laravel-crud/

https://www.studentstutorial.com/laravel/crud_example

https://www.studentstutorial.com/laravel/crud_example

CRUD Codeigniter

https://phpgurukul.com/crud-operation-in-codeigniter/

https://www.studentstutorial.com/codeigniter/codeigniter-crud_4

https://phpforever.com/codeigniterexample/codeigniter-4-crud-example/

https://www.positronx.io/codeigniter-crud-with-bootstrap-and-mysql-example/

What are the main features of Laravel?

Some of the main features of Laravel are:

- Eloquent ORM
- Query builder
- Reverse Routing
- Restful Controllers
- Migrations
- Database Seeding
- Unit Testing
- Homestead

What do you understand by Eloquent ORM?

Eloquent ORM (Object-Relational Mapping) is one of the main features of the Laravel framework. It may be defined as an advanced PHP implementation of the active record pattern.

Active record pattern is an architectural pattern which is found in software. It is responsible for keeping in-memory object data in relational databases

Eloquent ORM is also responsible for providing the internal methods at the same time when enforcing constraints on the relationship between database objects. Eloquent ORM represents database tables as classes, with their object instances tied to single table rows, while following the active record pattern.

What is Query Builder in Laravel?

Laravel's Query Builder provides more direct access to the database, alternative to the Eloquent ORM. It doesn't require SQL queries to be written directly. Instead, it offers a set of classes and methods which are capable of building queries programmatically. It also allows specific caching of the results of the executed queries.

Write down the name of some aggregates methods provided by the Laravel's query builder.

Some of the methods that Query Builder provides are:

- count()
- max()
- min()
- avg()
- sum()

What is routing?

All Laravel routes are defined in route files, which are stored in the routes directory. These files are loaded by the MVC framework. The routes/web.php files define routes that are available for the web interface. Those routes are allotted as the web middleware group, which provide features such as **session state** and **CSRF** protection. The routes available in **routes/api.php** are stateless and are allotted as the API middleware group. For most of the applications, one should start by defining routes in routes/web.php file.

What do you understand by Reverse routing?

Reverse routing in Laravel is used to generate the URL based on name or symbol. It defines a relationship between the links and, Laravel routes, and it is possible to make later changes to the routes to be automatically propagated into relevant links. When the links are generated using names of existing routes, the appropriate uniform resource identifiers (URIs) are automatically generated by Laravel. Reverse routing provides flexibility to the application and helps the developer to write cleaner codes.

Route Declaration:

```
1. Route::get('login', 'users@login');
```

A link can be created to it using reverse routing, which can be further transferred in any parameter that we have defined. If optional parameters are not supplied, they are removed automatically from the generated links.

```
1. {{ HTML::link_to_action('users@login') }}
```

By using it, a URL like https://abc.go.com/loginwill be created automatically.

How will you describe Bundles in Laravel?

In Laravel, Bundles are also known as **Packages**. Packages are the primary way to add more functionality to Laravel. Packages can be anything, from a great way to work with dates like Carbon, or an entire BDD testing framework like **Behat**. Laravel also provides support for creating custom packages.

There are different types of packages. Some of them are stand-alone packages. This means they can work with any PHP framework. The frameworks like **Carbon and Behat** are examples of stand-alone packages. Other packages are intended for use with Laravel. These packages may contain routes, controllers, views, and configurations which are mainly designed to enhance a Laravel application.

9) What is a composer, and how can we install Laravel by the composer?

A composer is a dependency manager in PHP. It manages the dependencies which are required for a project. It means that the composer will pull in all the necessary libraries, dependencies, and manage all at a single place.

Laravel Installation Steps:

- If you don't have a composer on a system, download composer from https://getcomposer.org/download/
- Open command prompt
- Go to htdocs folder
- Run the below command under C:\xampp\htdocs>

Does Laravel support caching?

Yes, Laravel provides support for popular caching backends like **Memcached** and **Redis**.

By default, Laravel is configured to use file cache driver, which is used to store the serialized or cached objects in the file system. For huge projects, it is suggested to use Memcached or Redis.

How to clear cache in Laravel?

The syntax to clear cache in Laravel is given below:

- php artisan cache: clear
- php artisan config: clear
- php artisan cache: clear

How will you explain middleware in Laravel?

As the name suggests, middleware works as a middleman between request and response. Middleware is a form of HTTP requests filtering mechanism. For example, Laravel consists of middleware which verifies whether the user of the application is authenticated or not. If a user is authenticated and trying to access the dashboard then, the middleware will redirect that user to home page; otherwise, a user will be redirected to the login page.

There are two types of middleware available in Laravel:

Global Middleware

It will run on every HTTP request of the application.

Route Middleware

It will be assigned to a specific route.

Syntax

1. php artisan make:middlewareMiddelwareName

Example

1. php artisan make:middlewareUserMiddleware

Now, UserMiddleware.php file will be created in app/Http/Middleware.

13) What do you understand by database migrations in Laravel? How can we use it?

Migrations can be defined as version control for the database, which allows us to modify and share the application's database schema easily. Migrations are commonly paired with **Laravel's schema builder** to build the application's database schema easily.

A migration file includes two methods, **up()** and **down()**. A method up() is used to add new tables, columns or indexes database and the down() method is used to reverse the operations performed by the up() method.

We can generate a migration and its file by using the **make:migration**.

Syntax

1. php artisan make:migration blog

By using it, a current date blog.php file will be created in database/migrations.

What do you know about Service providers in Laravel?

Service providers can be defined as the central place to configure all the entire Laravel applications. Applications, as well as Laravel's core services, are bootstrapped via service providers. These are powerful tools for maintaining class dependencies and performing dependency injection. Service providers also instruct Laravel to bind various components into the Laravel's Service Container.

An artisan command is given here which can be used to generate a service provider:

```
1. php artisan make: provider ClientsServiceProvider
```

Almost, all the service providers extend the Illuminate\Support\ServiceProviderclass. Most of the service providers contain below-listed functions in its file:

- Register() Function
- Boot() Function

Within the Register() method, one should only bind things into the service container. One should never attempt to register any event listeners, routes, or any other piece of functionality within the Register() method.

How can we get data between two dates using Query in Laravel?

We can use **whereBetween()** method to retrieve the data between two dates with Query.

Example

```
1. Blog::whereBetween('created_at', [$date1, $date2])->get();
```

What do you know about CSRF token in Laravel? How can someone turn off CSRF protection for a specific route?

CSRF protection stands for **Cross-Site Request Forgery** protection. CSRF detects unauthorized attacks on web applications by the unauthorized users of a system. The built-in CSRF plug-in is used to create CSRF tokens so that it can verify all the operations and requests sent by an active authenticated user.

To turn off CSRF protection for a specific route, we can add that specific URL or Route in $except variable which is present in the app\Http\Middleware\VerifyCsrfToken.phpfile.

Example

```
1. classVerifyCsrfToken extends BaseVerifier
2. {
3. protected $except = [
4.         'Pass here your URL',
5.     ];
6. }
```

List some official packages provided by Laravel?

There are some official packages provided by Laravel which are given below:

Cashier

Laravel cashier implements an expressive, fluent interface to Stripe's and Braintree's subscription billing services. It controls almost all of the boilerplate subscription billing code you are dreading writing. Moreover, the cashier can also control coupons, subscription quantities, swapping subscription, cancellation grace periods, and even generate invoice PDFs.

Envoy

Laravel Envoy is responsible for providing a clean, minimal syntax for defining frequent tasks that we run on our remote servers. Using Blade style syntax, one can quickly arrange tasks for deployment, Artisan commands, and more. Envoy only provides support for **Mac and Linux**.

Passport

Laravel is used to create API authentication to act as a breeze with the help of Laravel passport. It further provides a full **Oauth2** server implementation for Laravel application in a matter of minutes. Passport is usually assembled on top of **League OAuth2** server which is maintained by **Alex Bilbie**.

Scout

Laravel Scout is used for providing a simple, driver-based solution for adding full-text search to the eloquent models. Using model observers, Scout automatically keeps search indexes in sync with eloquent records.

Socialite

Laravel Socialite is used for providing an expressive, fluent interface to OAuth authentication with Facebook, Twitter, Google, and Linkedln, etc. It controls almost all the boilerplate social authentication code that you are dreading writing.

What do you understand by Unit testing?

Unit testing is built-in testing provided as an integral part of Laravel. It consists of unit tests which detect and prevent regressions in the framework. Unit tests can be run through the available **artisan** command-line utility.

What do you know about Facades in Laravel? Explain.

Laravel Facades provide static-like interface classes which are available in the application's service container. Laravel self-ships with several available facades, gives access to almost all features of Laravel. Facades also help to access a service directly from the container itself. It is described in the Illuminate\Support\Facades namespace. Hence, it is easy to use.

Example

```
1. use Illuminate\Support\Facades\Cache;
```

```
2.      Route::get('/cache', function () {
3.  return Cache::get('PutkeyNameHere');
4.  })
```

How can we check the Laravel current version?

One can easily check the current version of Laravel installation using the **-version** option of artisan command.

```
1.  Php artisan -version
```

How will you explain dd() function in Laravel?

dd stands for "**Dump and Die**." Laravel's dd() function can be defined as a helper function, which is used to dump a variable's contents to the browser and prevent the further script execution.

Example

```
1.  dd($array);
```

What do you know about PHP artisan? Mention some artisan command.

PHP artisan is a command-line interface/tool provided with Laravel. It consists of several useful commands which can be helpful while building an application. There are few artisan commands given below:

PHP artisan list

A 'list' command is used to view a list of all available Artisan commands.

PHP artisan help

Every command also contains a 'help' screen, which is used to display and describe the command's available arguments and options. To display a help screen, run 'help' command.

PHP artisan tinker

Laravel's artisan tinker is a repl (**Read-Eval-Print Loop**). Using tinker, one can write actual PHP code through command-line. One can even update or delete table records in the database.

PHP artisan -version

By using this command, one can view the current version of Laravel installation.

PHP artisan make model model_name

This command creates a model 'model_name.php' under the 'app' directory.

PHP artisan make controller controller_name

This command is used to build a new controller file in app/Http/Controllers folder.

How will you explain Events in Laravel?

An event is an activity or occurrence recognized and handled by the program. Events in Laravel provide simple observer implementations which allow us to subscribe and listen for events within our application. The event classes are stored in app/Events, while their listeners are stored in app/Listeners of our application. These can be generated using Artisan console commands. A single event may contain multiple listeners that do not depend on each other.

There are some events examples in Laravel which are:

- A new user is registered.
- A new comment is posted.
- User login/logout.
- A new product is added.

What are the validations in Laravel?

Validations are approaches that Laravel use to validate the incoming data within the application.

They are the handy way to ensure that data is in a clean and expected format before it gets entered into the database. Laravel consists of several different ways to validate the incoming data of the application. By default, the base controller class of Laravel uses a **ValidatesRequests** trait to validate all the incoming HTTP requests with the help of powerful validation rules.

What do you understand by Lumen?

Lumen is a PHP micro-framework built on Laravel's top components. It is created by **Taylor Otwell** (creator of Laravel). It is created for building Laravel based micro-services and blazing fast APIs. It is one of the fastest micro-frameworks available. Lumen is not a complete web framework like Laravel and used for creating APIs only. Therefore, most of the components, such as HTTP sessions, cookies, and templating, are excluded from Lumen. **Lumen** provides support for features such as logging, routing, caching queues, validation, error handling, middleware, dependency injection, controllers, blade templating, command scheduler, database abstraction, the service container, and Eloquent ORM, etc.

One can install Lumen using composer by running the command given below:

1. composer create-project --prefer-distlaravel/lumen blog

Which template engine is used by Laravel?

The **blade** is a simple but powerful templating engine provided with Laravel. There is no restriction to use PHP codes in the views. All the blade views are compiled into simple PHP code and cached until they are modified. Blade adds effectively zero overhead to our application. Blade view files in Laravel use the .blade.phpfile extension and are saved in the resources/views directory.

Explain the Service container and its advantages.

Service container in Laravel is one of the most powerful features. It is an important, powerful tool for resolving class dependencies and performing dependency injection in Laravel. It is also known as **IoC container**.

Dependency injection is a term which essentially means that class dependencies are "injected" into the class by the constructor or, in some cases," setter" methods.

Advantages of Service Container

- It can handle class dependencies on object creation.
- It can combine interfaces to concrete classes.

What do you know about Laravel Contracts?

Laravel's Contracts are the set of interfaces which are responsible for defining the core functionality of services provided by the Laravel framework.

How will you explain homestead in Laravel?

Homestead is an official, pre-packaged, vagrant virtual machine which provides Laravel developers all the necessary tools to develop Laravel out of the box. It also includes **Ubuntu, Gulp, Bower**, and other development tools which are useful in developing full-scale web applications. It provides a development environment which can be used without the additional need to install PHP, a web server, or any other server software on the machine.

What are the differences between Laravel and Codeigniter?

Laravel	Codeigniter
Laravel is a framework with an expressive, elegant syntax.	Codeigniter is a powerful framework based on PHP.
Laravel is built for the latest version of PHP.	Codeigniter is an older, more mature

	framework.
Laravel is more object-oriented as compared to Codeigniter.	Codeigniter is less object-oriented as compared to Laravel.
Laravel can produce model-view-controller, active-record, observer, dependency injection, singleton, restful, façade, event-driven, MTV, and HMVC design patterns.	Codeigniter can produce active-record, model-view-controller, and HMVC design patterns.
Laravel supports ORM.	Codeigniter does not support ORM
Laravel needs 1 GB memory.	Codeigniter needs 256 GB memory.
Laravel has built-in user authentication support.	Codeigniter does not have in-built user authentication support.

How can we get the user's IP address in Laravel?

We can get the user's IP address using:

```
1. public function getUserIp(Request $request){
2. // Gettingip address of remote user
3. return $user_ip_address=$request->ip();
4. }
```

How can we use the custom table in Laravel?

We can easily use custom table in Laravel by overriding protected $table property of Eloquent. Here, is the sample:

```
1. class User extends Eloquent{
2. protected $table="my_user_table";
3. }
```

What is the use of the Eloquent cursor() method in Laravel?

The cursor method allows us to iterate through our database using a cursor, which will only execute a single query. While processing large amounts of data, the cursor method may be used to reduce your memory usage greatly.

Example

```
1. foreach (Product::where('name', 'bar')->cursor() as $flight) {
2. //make some stuff
3. }
```

How will you create a helper file in Laravel?

We can create a helper file using composer as per the given below steps:

Make a file "app/helpers.php" within the app folder.

Add

```
"files": [
"app/helpers.php"
]
```

in the "autoload" variable.

Now update composer.json with composer dump-autoload or composer update.

What are the requirements for Laravel 5.8?

- PHP Version>=7.1.3
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension
- Ctype PHP Extension
- JSON PHP Extension

In which directory controllers are kept in Laravel?

Controllers are kept in **app/http/Controllers** directory.

What is the use of PHP compact function?

PHP compact function receives each key and tries to search a variable with that same name. If a variable is found, then it builds an associate array.

Explain some benefits of Laravel over other PHP frameworks.

There are few benefits of Laravel which can be considered over other PHP frameworks:

- In Laravel, Setup and customization process is fast and easy as compared to others.
- Laravel supports multiple file systems.
- It has pre-loaded packages like **Laravel Socialite, Laravel cashier, Laravel Passport, Laravel elixir, and Laravel Scout**, etc.
- It consists of in-built Authentication System.
- It supports Eloquent ORM (Object Relation Mapping) with PHP active record implementation.
- "Artisan" command-line tool for creating a database structure, code skeleton, and build their migration.

Which types of relationships are available in Laravel Eloquent?

Below are the types of relationships that Laravel Eloquent ORM supports:

- One to One
- One to Many
- One to Many (Inverse)
- Many to Many
- Has Many Through
- Polymorphic Relations
- Many To Many Polymorphic Relations

What do you understand by ORM?

ORM stands for **Object-Relational Mapping**. It is a programming technique which is used to convert data between incompatible type systems in object-oriented programming languages.

How can we implement a package in Laravel?

We can implement a package in Laravel by:

- Creating a package folder and name it.
- Creating **Composer.json** file for the package.
- Loading package through main composer.json and PSR-4.
- Creating a Service Provider.
- Creating a Controller for the package.
- Creating a Routes.php file.

What do you know about Traits in Laravel?

PHP Traits is a group of methods which can be included within another class. A Trait cannot be instantiated by itself like an abstract class. Traits are generated to reduce the limitations of single inheritance in PHP. It allows a developer to reuse sets of methods freely in various independent classes living in different class hierarchies.

Example

```
1. trait Sharable {
2. public function share($item)
3.  {
4. return 'share this item';
5.  }
6. }
```

We can then include this Trait within other classes like:

```
1. class Post {
2. use Sharable;
3. }
4. class Comment {
5. use Sharable;
6. }
```

Now, if we want to create new objects out of these classes, we would find that they both have the share() method available:

```
1. $post = new Post;
2. echo $post->share(''); // 'share this item'
3. $comment = new Comment;
4. echo $comment->share(''); // 'share this item'
```

How can someone change the default database type in Laravel?

Laravel is configured to use MySQL by default.

To change its default database type, edit the file **config/database.php**:

- Search for 'default' =>env('DB_CONNECTION', 'mysql')
- Change it to whatever required like 'default' =>env('DB_CONNECTION', 'sqlite')

By using it, MySQL changes to SQLite.

How can we use maintenance mode in Laravel 5?

When an application is in maintenance mode, a custom view is displayed for all requests into the application. It makes it easy to "disable" application while it is updating or performing maintenance. A maintenance mode check is added in the default middleware

stack for our application. When an application is in maintenance mode, a **MaintenanceModeException** will be thrown with a status code of 503.

We can enable or disable maintenance mode in Laravel 5, simply by executing the below command:

```
// Enable maintenance mode
php artisan down

// Disable maintenance mode
php artisan up
```

How can we create a record in Laravel using eloquent?

We need to create a new model instance if we want to create a new record in the database using Laravel eloquent. Then we are required to set attributes on the model and call the save() method.

Example

```
public functionsaveProduct(Request $request )
$product = new product;
$product->name = $request->name;
$product->description = $request->name;
$product->save();
```

How can we check the logged-in user info in Laravel?

User() function is used to get the logged-in user

Example

```
if(Auth::check()){
$loggedIn_user=Auth::User();
dd($loggedIn_user);
}
```

How will you describe Fillable Attribute in a Laravel model?

In eloquent ORM, $fillable attribute is an array containing all those fields of table which can be filled using mass-assignment.

Mass assignment refers to sending an array to the model to directly create a new record in Database.

Code Source

```
class User extends Model {
protected $fillable = ['name', 'email', 'mobile'];
```

```
3.  // All fields inside $fillable array can be mass-assigned
4.  }
```

How will you explain Guarded Attribute in a Laravel model?

The guarded attribute is the opposite of fillable attributes.

In Laravel, fillable attributes are used to specify those fields which are to be mass assigned. Guarded attributes are used to specify those fields which are not mass assignable.

Code Source

```
1.  class User extends Model {
2.  protected $guarded = ['role'];
3.  // All fields inside the $guarded array are not mass-assignable
4.  }
```

If we want to block all the fields from being mass-assigned, we can use:

```
1.  protected $guarded = ['*'];
```

$fillable serves as a "white list" whereas $guarded functions serves like a "black list". One should use either $fillable or $guarded.

What do you know about Closures in Laravel?

In Laravel, a Closure is an anonymous method which can be used as a **callback** function. It can also be used as a parameter in a function. It is possible to pass parameters into a Closure. It can be done by changing the Closure function call in the **handle()** method to provide parameters to it. A Closure can access the variables outside the scope of the variable.

Define composer.

It is an application-level package manager for PHP. It provides a standard format for managing PHP software dependencies and libraries.

What is HTTP middleware?

HTTP middleware is a technique for filtering HTTP requests. Laravel includes a middleware that checks whether application user is authenticated or not.

Name aggregates methods of query builder.

Aggregates methods of query builder are: 1) max(), 2) min(), 3) sum(), 4) avg(), and 5) count().

What is a Route?

A route is basically an endpoint specified by a URI (Uniform Resource Identifier). It acts as a pointer in Laravel application.

Most commonly, a route simply points to a method on a controller and also dictates which HTTP methods are able to hit that URI.

Why use Route?

Routes are stored inside files under the /routes folder inside the project's root directory. By default, there are a few different files corresponding to the different "sides" of the application ("sides" comes from the hexagonal architecture methodology).

What do you mean by bundles?

In Laravel, bundles are referred to as packages. These packages are used to increase the functionality of Laravel. A package can have views, configuration, migrations, routes, and tasks.

Explain important directories used in a common Laravel application.

Directories used in a common Laravel application are:

- App/: This is a source folder where our application code lives. All controllers, policies, and models are inside this folder.
- Config/: Holds the app's configuration files. These are usually not modified directly but instead, rely on the values set up in the .env (environment) file at the root of the app.
- Database/: Houses the database files, including migrations, seeds, and test factories.
- Public/: Publicly accessible folder holding compiled assets and of course an index.php file.

What is a Controller?

A controller is the "C" in the "MVC" (Model-View-Controller) architecture, which is what Laravel is based on.

Explain reverse routing in Laravel.

Reverse routing is a method of generating URL based on symbol or name. It makes your Laravel application flexible.

Explain traits in Laravel.

Laravel traits are a group of functions that you include within another class. A trait is like an abstract class. You cannot instantiate directly, but its methods can be used in concreate class.

Explain the concept of contracts in Laravel.

They are set of interfaces of Laravel framework. These contracts provide core services. Contracts defined in Laravel include corresponding implementation of framework.

How will you register service providers?

You can register service providers in the config/app.php configuration file that contains an array where you can mention the service provider class name.

Where will you define Laravel's Facades?

All facades of Laravel have defined in Illuminate\Support\Facades namespace.

State the difference between get and post method.

Get method allows you to send a limited amount of data in the header. Post allows you to send a large amount of data in the body.

What is service container in Laravel?

Service container is a tool used for performing dependency injection in Laravel.

How can you enable query log in Laravel?

You can use enableQueryLog method to enable query log in Laravel.

Explain the concept of events in Laravel.

An event is an occurrence or action that help you to subscribe and listen for events that occur in Laravel application. Some of the events are fired automatically by Laravel when any activity occurs.

Explain dependency injection and their types.

It is a technique in which one object is dependent on another object. There are three types of dependency injection: 1) Constructor injection, 2) setter injection, and 3) interface injection.

What are the advantages of using Laravel?

Here are important benefits of Laravel:

- Laravel has blade template engine to create dynamic layouts and increase compiling tasks.
- Reuse code without any hassle.
- Laravel provides you to enforce constraints between multiple DBM objects by using an advanced query builder mechanism.
- The framework has an auto-loading feature, so you don't do manual maintenance and inclusion paths

- The framework helps you to make new tools by using LOC container.
- Laravel offers a version control system that helps with simplified management of migrations.

Explain validation concept in Laravel.

Validations are an important concept while designing any Laravel application. It ensures that the data is always in an expected format before it stores into the database. Laravel provides many ways to validate your data.

Base controller trait uses a ValidatesRequests class which provides a useful method to validate requests coming from the client machine.

What does ORM stand for?

ORM stands for Object Relational Mapping

How can you reduce memory usage in Laravel?

While processing a large amount of data, you can use the cursor method in order to reduce memory usage.

List available types of relationships in Laravel Eloquent.

Types of relationship in Laravel Eloquent are: 1) One To One 2) One To Many 3) Many To Many 4) Has Many Through, and 5) Polymorphic Relations.

Name the Template Engine utilized by Laravel.

Blade is a powerful template engine utilized by Laravel.

Name databases supported by Laravel.

Laravel supports the following databases:

- PostgreSQL
- SQL Server
- SQLite
- MySQL

Why are migrations important?

Migrations are important because it allows you to share application by maintaining database consistency. Without migration, it is difficult to share any Laravel application. It also allows you to sync database.

Define Lumen

Lumen is a micro-framework. It is a smaller, and faster, version of a building Laravel based services, and REST API's.

Explain PHP artisan

An artisan is a command-line tool of Laravel. It provides commands that help you to build Laravel application without any hassle.

How can you generate URLs?

Laravel has helpers to generate URLs. This is helpful when you build link in your templates and API response.

Which class is used to handle exceptions?

Laravel exceptions are handled by App\Exceptions\Handler class.

What are common HTTP error codes?

The most common HTTP error codes are:

- Error 404 – Displays when Page is not found.
- Error- 401 – Displays when an error is not authorized

Explain fluent query builder in Laravel.

It is a database query builder that provides convenient, faster interface to create and run database queries.

What is the use of dd() function?

This function is used to dump contents of a variable to the browser. The full form of dd is Dump and Die.

List out common artisan commands used in Laravel.

Laravel supports following artisan commands:

- PHP artisan down;
- PHP artisan up;
- PHP artisan make:controller;
- PHP artisan make:model;
- PHP artisan make:migration;
- PHP artisan make:middleware;

How to configure a mail-in Laravel?

Laravel provides APIs to send an email on local and live server.

Explain Auth.

It is a method of identifying user login credential with a password. In Laravel it can be managed with a session which takes two parameters 1) username and 2) password.

Differentiate between delete() and softDeletes().

- delete(): remove all record from the database table.
- softDeletes(): It does not remove the data from the table. It is used to flag any record as deleted.

How can you make real time sitemap.xml file in Laravel?

You can create all web pages of a website to tell the search engine about the organizing site content. The crawlers of search engine read this file intelligently to crawl a website.

Explain faker in Laravel.

It is a type of module or packages which are used to create fake data. This data can be used for testing purpose.

It is can also be used to generate: 1) Numbers, 2) Addresses, 3) DateTime, 4) Payments, and 5) Lorem text.

How will you check table is exists or in the database?

Use hasTable() Laravel function to check the desired table is exists in the database or not.

What is the significant difference between insert() and insertGetId() function in Laravel?

- Insert(): This function is simply used to insert a record into the database. It not necessary that ID should be autoincremented.
- InsertGetId(): This function also inserts a record into the table, but it is used when the ID field is auto-increment.

Explain active record concept in Laravel.

In active record, class map to your database table. It helps you to deal with CRUD operation.

List basic concepts in Laravel?

Following are basic concepts used in Laravel:

- Routing
- Eloquent ORM
- Middleware
- Security
- Caching
- Blade Templating

Define Implicit Controller.

Implicit Controllers help you to define a proper route to handle controller action. You can define them in route.php file with Route:: controller() method.

How to use the custom table in Laravel Model?

In order to use a custom table, you can override the property of the protected variable $table.

What is MVC framework?

It is Model, View, and Controller:

- Model: Model defines logic to write Laravel application.
- View: It covers UI logic of Laravel application.
- Controller: It is work as an interface between Model, and View. It is a way how the user interacts with an application.

50) Define @include.

@include is used to load more than one template view files. It helps you to include view within another view. User can also load multiple files in one view.

51) Explain the concept of cookies.

Cookies are small file sent from a particular website and stored on PC by user's browser while the user is browsing.

Which file is used to create a connection with the database?

To create a connection with the database, you can use .env file.

What is Eloquent?

Eloquent is an ORM used in Laravel. It provides simple active record implementation working with the database. Each database table has its Model, which used to interact with the table.

Name some Inbuilt Authentication Controllers of Laravel.

Laravel installation has an inbuilt set of common authentication controllers. These controllers are:

- RegisterController
- LoginController
- ResetPasswordController
- ForgetPasswordController

Define Laravel guard.

Laravel guard is a special component that is used to find authenticated users. The incoming requested is initially routed through this guard to validate credentials entered by users.

What is Laravel API rate limit?

It is a feature of Laravel. It provides handle throttling. Rate limiting helps Laravel developers to develop a secure application and prevent DOS attacks.

Explain collections in Laravel.

Collections is a wrapper class to work with arrays. Laravel Eloquent queries use a set of the most common functions to return database result.

What is the use of DB facade?

DB facade is used to run SQL queries like create, select, update, insert, and delete.

What is the use of Object Relational Mapping?

Object Relational Mapping is a technique that helps developers to address, access, and manipulate objects without considering the relation between object and their data sources.

Explain the concept of routing in Laravel.

It allows routing all your application requests to the controller. Laravel routing acknowledges and accepts a Uniform Resource Identifier with a closure.

What is Ajax in Laravel?

Ajax stands for Asynchronous JavaScript and XML is a web development technique that is used to create asynchronous Web applications. In Laravel, response() and json() functions are used to create asynchronous web applications.

What is a session in Laravel?

Session is used to pass user information from one web page to another. Laravel provides various drivers like a cookie, array, file, Memcached, and Redis to handle session data.

How to access session data?

Session data be access by creating an instance of the session in HTTP request. Once you get the instance, use get() method with a “Key” as a parameter to get the session details.

State the difference between authentication and authorization.

Authentication means confirming user identities through credentials, while authorization refers to gathering access to the system.

Explain to listeners.

Listeners are used to handling events and exceptions. The most common listener in Laravel for login event is LoginListener.

What are policies classes?

Policies classes include authorization logic of Laravel application. These classes are used for a particular model or resource.

How to rollback last migration?

Use need to use artisan command to rollback the last migration.

What do you mean by Laravel Dusk?

Laravel Dusk is a tool which is used for testing JavaScript enabled applications. It provides powerful, browser automation, and testing API.

Explain Laravel echo.

It is a JavaScript library that makes possible to subscribe and listen to channels Laravel events. You can use NPM package manager to install echo.

What is make method?

Laravel developers can use make method to bind an interface to concreate class. This method returns an instance of the class or interface. Laravel automatically inject dependencies defined in class constructor.

Explain Response in Laravel.

All controllers and routes should return a response to be sent back to the web browser. Laravel provides various ways to return this response. The most basic response is returning a string from controller or route.

What is query scope?

It is a feature of Laravel where we can reuse similar queries. We do not require to write the same types of queries again in the Laravel project. Once the scope is defined, just call the scope method when querying the model.

Explain homestead in Laravel.

Laravel homestead is the official, disposable, and pre-packaged vagrant box that a powerful development environment without installing HHVM, a web server, and PHP on your computer.

What is namespace in Laravel?

A namespace allows a user to group the functions, classes, and constants under a specific name.

What is Laravel Forge?

Laravel Forge helps in organizing and designing a web application. Although the manufacturers of the Laravel framework developed this toll, it can automate the deployment of every web application that works on a PHP server.

State the difference between CodeIgniter and Laravel.

Parameter	CodeIgniter	Laravel
Support of ORM	CodeIgniter does not support Object-relational mapping.	Laravel supports ORM.
Provide Authentication	It does provide user authentication.	It has inbuilt user authentication.
Programming Paradigm	It is component-oriented.	It is object-oriented.
Support of other Database Management System	It supports Microsoft SQL Server, ORACLE, MYSQL, IBM DB2, PostgreSQL, JDBC, and orientDB compatible.	It supports PostgreSQL, MySQL, MongoDB, and Microsoft BI, but CodeIgniter additionally supports other databases like Microsoft SQL Server, DB2, Oracle, etc.
HTTPS Support	CodeIgniter partially support HTTPS. Therefore, programmers can use the URL to secure the data transmission process by creating PATS.	Laravel supports custom HTTPS routes. The programmers can create a specific URL for HTTPS route they have defined.

What is an Observer?

Model Observers is a feature of Laravel. It is used to make clusters of event listeners for a model. Method names of these classes depict the Eloquent event. Observers classes methods receive the model as an argument.

What is the use of the bootstrap directory?

It is used to initialize a Laravel project. This bootstrap directory contains app.php file that is responsible for bootstrapping the framework.

What is the default session timeout duration?

The default Laravel session timeout duration is 2 hours.

How to remove a complied class file?

Use clear-compiled command to remove the compiled class file.

In which folder robot.txt is placed?

Robot.txt file is placed in Public directory.

Explain API.PHP route.

Its routes correspond to an API cluster. It has API middleware which is enabled by default in Laravel. These routes do not have any state and cross-request memory or have no sessions.

What is named route?

Name route is a method generating routing path. The chaining of these routes can be selected by applying the name method onto the description of route.

what is open source software?

Open-source software is a software which source code is freely available. The source code can be shared and modified according to the user requirement.

Explain Loggin in Laravel.

It is a technique in which system log generated errors. Loggin is helpful to increase the reliability of the system. Laravel supports various logging modes like syslog, daily, single, and error log modes.

What is Localization?

It is a feature of Laravel that supports various language to be used in the application. A developer can store strings of different languages in a file, and these files are stored at resources/views folder. Developers should create a separate folder for each supported language.

Define hashing in Laravel.

It is the method of converting text into a key that shows the original text. Laravel uses the Hash facade to store the password securely in a hashed manner.

Explain the concept of encryption and decryption in Laravel.

It is a process of transforming any message using some algorithms in such way that the third user cannot read information. Encryption is quite helpful to protect your sensitive information from an intruder.

Encryption is performed using a Cryptography process. The message which is to be encrypted called as a plain message. The message obtained after the encryption is referred to as cipher message. When you convert cipher text to plain text or message, this process is called as decryption.

How to share data with views?

To pass data to all views in Laravel use method called share(). This method takes two arguments, key, and value.

Generally, share() method are called from boot method of Laravel application service provider. A developer can use any service provider, AppServiceProvider, or our own service provider.

Explain web.php route.

Web.php is the public-facing “browser” based route. This route is the most common and is what gets hit by the web browser. They run through the web middleware group and also contain facilities for CSRF protection (which helps defend against form-based malicious attacks and hacks) and generally contain a degree of “state” (by this I mean they utilize sessions).

How to generate a request in Laravel?

Use the following artisan command in Laravel to generate request:

```
php artisan make:request UploadFileRequest
```

These interview questions will also help in your viva(orals)

Why is there a need to configure the URL routes?

Changing the URL routes has many benefits such as:

- From the SEO(Search Engine Optimization) point of the view, to make URL SEO friendly and obtain more user visits.
- Hide some URL elements like controller name, function name, etc. from the users for security purposes.
- Provides different functionality to the specific parts of a system.

CodeIgniter

What are the most prominent features of CodeIgniter?

A list of most prominent features of CodeIgniter:

- It is an open source framework and free to use.
- It is extremely light weighted.
- It is based on the Model View Controller (MVC) pattern.
- It has full featured database classes and support for several platforms.
- It is extensible. You can easily extend the system by using your libraries, helpers.
- Excellent documentation.

Explain the folder structure of CodeIgniter.

If you download and unzip CodeIgniter, you get the following file structure/folder structure:

Application

- cache
- Config
- Controllers
- core
- errors
- helpers
- hooks
- language
- libraries
- logs
- models
- third-party
- views

system

- core
- database
- fonts
- helpers
- language
- libraries

Explain CodeIgniter architecture.

From a technical point of view, CodeIgniter is dynamically instantiation (light-weighted), loosely coupled (components rely very less on each other) and has a component singularity (each class and functions are narrowly focused towards their purpose).

How can you add or load a model in CodeIgniter?

To load models in controller functions, use the following function:

```
1. $this->load->model('ModelName');
```

If in case your model file is located in sub-directory of the model folder, then you have to mention the full path. For example, if your file location is application/controller/models/project/ModelName. Then, your file will be loaded as shown below,

```
1. $this->load->model('project/ModelName');
```

How can you connect models to a database manually?

To connect database manually use following syntax,

```
1. $this->load->database();
```

Explain views in CodeIgniter.

View folder contains all the markup files like header, footer, sidebar, etc. They can be reused by embedding them anywhere in controller file. They can't call directly, and they have to be loaded in the controller's file.

View syntax

Create a file and save it in application/views folder. For example, we have created a file Firstview.php,

How can you load a view in CodeIgniter?

The View can't be accessed directly. It is always loaded in the controller file. Following function is used to load a view page:

```
1. $this->load->view('page_name');
```

Write your view's page name in the bracket. You don't need to specify **.php** unless you are using some other extension.

Now, go to your controller file **(Main.php)** and write this code as shown below.

Explain controller in CodeIgniter.

A controller is the intermediary between models and views to process the HTTP request and generates a web page. It is the center of every request on your web application.

Consider following URI,

```
1. abc.com/index.php/front/
```

In this URI, CodeIgniter try to find Front.php file and Front class.

Controller Syntax

Look at the above snapshot, controller's file name is **Main.php** (the first letter has to be in uppercase), and the class name is **Main** (the first letter has to be in uppercase).

What is the default controller in CodeIgniter?

The file specified in the default controller loaded by default when no file name is mentioned in the URL. By default, it is welcome.php which is the first page to be seen after installing CodeIgniter.

With URL

```
1. localhost/codeigniter/
```

Welcome.php will be loaded as there is no file name mentioned in the URL.

Although as per your need, you can change the default controller in the file **application/config/routes.php.**

```
1. $route['default_controller'] = ' ';
```

Here, specify your file name which you want to be loaded by default.

How will you call a constructor in CodeIgniter?

To use a constructor, you need to mention the following line of code,

```
1. parent::_construct()
```

What is an inhibitor of CodeIgniter?

In CodeIgniter, Inhibitor is an error handler class that uses native PHP functions like set_exception_handler, set_error_handler, register_shutdown_function to handle parse errors, exceptions, and fatal errors.

What is the default method name in CodeIgniter?

By default controller always calls index method. If you want to call a different method, then write it in the controller?s file and specify its name while calling the function.

Explain the remapping method calls in CodeIgniter.

The Second segment of URI determines which method is being called. If you want to override it, you can use _remap() method. The _remap method always get called even if URI is different. It overrides the URI. For Example:

```
public function _remap($methodName)
{
        if ($methodName === 'a_method')
        {
        $this->method();
        }
        else
        {
        $this->defaultMethod();
        }
}
```

What is a helper in CodeIgniter? How can a helper file be loaded?

Helpers are the group of functions that are used to assist the user to perform specific tasks.

URL Helpers: used to create the links.

Text Helpers: used for text formatting.

Cookies Helpers: used for reading and setting cookies.

How can you load multiple helper files?

To load multiple helper files, specify them in an array,

```
$this->load->helper(
array('helper1', 'helper2', 'helper3')
);
```

Explain the CodeIgniter library. How will you load it?

CodeIgniter provides a rich set of libraries. It is an essential part of CodeIgniter as it increases the developing speed of an application. It is located in the system/library.

It can be loaded as follows,

```
$this->load->library('class_name');
```

How can you create a library in CodeIgniter?

There are three methods to create a library,

- Creating an entirely new library
- Extending native libraries
- Replacing native libraries

Where is a newly created library stored in CodeIgniter structure?

It should be placed in application/libraries folder.

Can you extend native libraries in CodeIgniter?

Yes, we can add some extended functionality to a native library by adding one or two methods. It replaces the entire library with your version. So it is better to extend the class. Extending and replacing is almost identical with only following exceptions.

- The class declaration must extend the parent class.
- New class name and filename must be prefixed with **MY_.**

For example, to extend it to native Calendar, create a file **MY_Calendar.php** in **application/libraries** folder. Your class declared as **class MY_Calendar extends CI_Calendar}**

How can you extend a class in CodeIgniter?

You have to build a file name application/core/MY_Input.php and declare your class with **Class MY_Input extends CI_Input {}**to extend the native input class in CodeIgniter.

What is routing in CodeIgniter?

Routing is a technique by which you can define your URLs according to the requirement instead of using the predefined URLs. Routes can be classified in two ways, either using Wildcards or Regular Expressions.

Wildcards

There are two types of wildcards:

- :num−series containing only numbers matched.
- :any−series containing only characters matched.

Regular Expression

Regular expressions are also used to redirect routes.

1. $route['blog'(a-zA-Z0-9]+)'] = 'women/social';

You can create your regular expression to run your URL.

Why is URL routes need to be configured?

There are many purposes for which the URL routes are configured.

1. To improve the number of page visits.
2. To hide the code complexities from the user.

What are the hooks in CodeIgniter?

The Hook is a feature in CodeIgniter that provides a way to change the inner working of the framework without hacking the core files. It facilitates you to execute a script with a particular path within the CodeIgniter. Usually, it is defined in the **application/config/hooks.php** file.

How to enable CodeIgniter hook?

To enable hook, go to **application/config/config.php/** file and set it TRUE as shown below,

```
1. $config['enable_hooks'] = TRUE;
```

What are different types of hook points in CodeIgniter?

A list of different types of hook points in CodeIgniter:

- post_controller_constructor - It is called immediately after your controller is started but before any method call.
- pre_controller - It is called immediately before your controller being called. At this point, all the classes, security checks, and routing have been done.
- post_sytem - It is called after the final page is sent to the browser at the end of the system execution.
- pre_system - It is called much before the system execution. Only benchmark and hook class have been loaded at this point.
- cache_override - It enables you to call your function in the output class.
- display_override - It is used to send the final page at the end of file execution.
- post_controller - It is called immediately after your controller is entirely executed.

What are CodeIgniter drivers?

These are a particular type of library that has a parent class and many child classes. These child classes have access to the parent class, but not to their siblings. Drivers are found in **system/libraries** folder.

How to initialize a driver in CodeIgniter?

To initialize a driver, write the following syntax,

```
1. $this->load->driver('class_name');
```

Here, class_name is the driver name.

How to create a driver in CodeIgniter?

There are three steps to create a driver:

1. Making file structure
2. Making driver list
3. Making driver(s)

How to connect multiple databases in CodeIgniter?

To connect more than one database simultaneously, do the following,

```
$db1 = $this->load->database('group_one', TRUE);
$db1 = $this->load->database('group_two', TRUE);
```

How can you print SQL statement in CodeIgniter model?

```
$this>db>insertid();
```

What are CodeIgniter security methods?

CodeIgniter security methods help to create a secure application and process input data. The methods are given below:

- XSS filtering
- CSRF (Cross-site Request Forgery)
- Class reference

What are the XSS security parameters?

XSS stands for cross-site scripting. Codeigniter contains a cross-site scripting hack prevention filter. The XSS filter targets methods to trigger JavaScript or other types of suspicious code. If it detects anything, it converts the data to character entities.

XSS filtering uses xss_clean() method to filer data.

```
$data = $this->security->xss_clean($data);
```

There is an optional second parameter, is_image, which is used to test images for XSS attacks. When this parameter is set to TRUE, it doesn't return an altered string. Instead, it returns TRUE if an image is safe and FALSE if it contains malicious information.

```
if ($this->security->xss_clean($file, TRUE) === FALSE)
    {
        //file failed in xss test
    }
```

How can the CodeIgniter be prevented from CSRF?

There are the various ways by which, we can prevent CodeIgniter from CSRF. The most used method is using the hidden field in each page of the website. The hidden field is stored in the user's session. The filed is changed with every HTTP request. The user can be detected in its every request to the website. The hidden value is always compared with the one saved in the session. If it is the same, the request is valid.

How can you enable CSRF?

You can enable protection by editing config.php file and setting it to

To enable CSRF make the following statement TRUE from FALSE in **application/config/config.php** file.

```
1. $config['csrf_protection'] = TRUE;
```

More Details.

What is CSRF attack in CodeIgniter?

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including victim's session cookie and other authentication information, to a web application.

For example, suppose you have a site with a form. An attacker could create a bogus form on his site. This form could contain hidden inputs and malicious data. This form is not sent to the attacker's site, in fact, it comes to your site. Thinking that the form is genuine, your site process it.

Now suppose that the attacker's form point towards the deletion form in your site. If a user is logged in and redirected to the attacker's site and then perform the search, his account will be deleted without knowing him. That is the CSRF attack.

What is a token method in a CSRF attack?

To protect from CSRF, we need to connect both HTTP requests, form request and form submission. There are several ways to do this, but in CodeIgniter hidden field is used which is called the CSRF token. The CSRF token is a random value that changes with every HTTP request sent.

With each request, a new CSRF token is generated. When an object is created, name and value of the token are set.

```
1. $this->csrf_cookie_name = $this->csrf_token_name;
2. $this->_csrf_set_hash();
```

The function for it is,

```
1. function _csrf_set_hash()
2. {
3.     if ($this->csrf_hash == '')
4.     {
5. if ( isset($_COOKIE[$this->csrf_cookie_name] ) AND
```

```
6.              $_COOKIE[$this->csrf_cookie_name] != '' )
7.            {
8.              $this->csrf_hash = $_COOKIE[$this->csrf_cookie_name];
9.            } else {
10.               $this->csrf_hash = md5(uniqid(rand(), TRUE));
11.           }
12.         }
13.     return $this->csrf_hash;
```

What are the features of CodeIgniter ?

1. It is an open source framework and free to use.
2. It is extremely light weighted.

3. It is based on MVC(Model, View, Controller) pattern.

4.It has full featured database classes and support for several platforms.

5.Excellent documentation.

How you can add or load a model in CodeIgniter ?

```
$this->load->model('Model_Name');
```

How you can connect models to database manually ?

To connect database manually use following syntax :

```
$this>load->database();
```

Explain views in CodeIgniter ?

Answer : View folder contains all the markup files like header,footer, sidebar etc. They can be reused by embedding them anywhere in the controller file. They can't called directly, they have to be loaded in the controller file.

How can you load a view in CodeIgniter ?

```
$this->load->view('Page_Name');
```

Explain controller in CodeIgniter ?

A Controller is the intermediate between models and view to process HTTP request and generates a web page. It is the center of every request on your web application.

What is the default controller in CodeIgniter ?

The file specified in default controller will be loaded by default .When no file name is mentioned in the URL. By default, it is welcome.php which is the first page to be seen after installing CodeIgniter.

What is the basic CodeIginter URL structure ?

Instead of using query-string approach, it uses a segment base approach.
It's structure is as follows :

xyz.com/class/function/Id

Class represent controller class that need to be invoked.

Function is the method that is called.

Id is any additional segment that is passed to controllers.

How will you call a constructor in CodeIgniter ?

parent::_construct();

Explain what are hooks in CodeIgniter ?

CodeIgniter's hooks feature provide a way to change the inner working of framework without hacking the core files. In other words, hooks allow you to execute a script with a particular path with in the CodeIgniter. Usually, it is defined in application/config/hooks.php file.
The hoks feature can be globally enabled/disabled by setting the following item in the application/config/config.php file :

$config['enablehooks']=TRUE;

Explain what helpers in CodeIgniter are and how you can load a helper file ?

In CodeIgniter, helpers are group of function in a particular category that assist you to perform specific functions. In CodeIgniter, you will find many helpers like URL helpers- helping in creating links, Text helpers- perform various text formatting routines, Cookies- helpers set and read cookies. You can can load helper file using command
$this->load->helper('name');

List out different types of hook point in CodeIgniter ?

Different types of hook point in CodeIgniter includes
post_controller_constructor

pre_controller

post_system

pre_system

cache_override

display_override

post_controller

$hook[‘pre_controller’]=array(

‘class’=>’MyClass’,

‘function’=>’MyFunction’,

‘filename’=>’Myclass.php’,

‘filepath’=>’hooks’,

‘params’=>array(‘param1′,’param2′,’param3’)

);

Explain what is inhibitor in CodeIgniter ?

inhibitor is an error handler class, using the native PHP functions like set_exception_hnadler, set_error_handler, register_shutdown_function to handle parse errors, exceptions, and fatal errors.

Explain how you can extend the class in CodeIgniter ?

To extend the native input class in CodeIgniter, you have to build a file named application/core/MY_Input.php and declare your class with

```
1 Class MY_Input extends CI_Input
2 {
3 }
```

Explain how you can enable CSRF in CodeIgniter ?

You can activate CSRF (Cross Site Request Forgery) protection in CodeIgniter by operating application/config/config.php file and setting it to $config[‘csrf_protection’]=TRUE; If the form helper, the from_open() function will insert a hidden csrf field in your forms automatically.

How to access config variable in CodeIgniter ?

```
$this->config->item('variable_name');
```

How to unset session in CodeIgniter ?

```
1 $this->session->unsetuserdata('somename');
```

How do you get last insert id in CodeIgniter ?

$this->db->insertid();

What are hooks in CodeIgnitor?

- CodeIgniter's hooks will provide a way to change the internal workings or framework functionalities without any need for hacking the core files. It permits script execution with a particular path within the CodeIgniter.
- We can globally enable/disable the hooks feature by setting the below-given item in the `application/config/config.php` file: `$config['enable_hooks'] = TRUE;`
- It is defined in `application/config/hooks.php` file. For example:

```
$hook['pre_controller'] = array(
'class' => 'MyHookClass',
'function' => 'Myhookfunction',
'filename' => 'MyHookClass.php',
'filepath' => 'hooks',
'params' => array('test', 'test1', 'webs')
);
```

In the above code example, the 'pre_controller' hook is called hook point. Various types of hook points are available in CodeIgniter.

What is an inhibitor in CodeIgniter?

An inhibitor in CodeIgniter is an error handler class. It will make use of PHP's native functions like `set_error_handler`, `set_exception_handler`, `register_shutdown_function` to handle parse errors, exceptions, and fatal errors.

How to check the CodeIgniter version?

There are 2 ways to check the CodeIgniter version.

- The first method is to run the following code:

```
<?php
   echo CI_VERSION;
?>
```

You can echo the constant value of CI_VERSION in the CodeIgniter controller or view file.

- The second method is to navigate to the `system/core/CodeIgniter.php` directory which stores the current version number of CodeIgniter in a global constant named 'CI_VERSION'. Open the file and have a look at the lines:

```
/**
 * CodeIgniter Version
 *
 * @var    string
 *
 */
  define('CI_VERSION', '4.1.3');
```

Explain the difference between helper and library in CodeIgniter.

Helper	Library
Helper is a collection of common functions which we can use within Models, Views as well as in Controllers. Once we include the helper file, we can get access to the functions.	Library is a class that has a set of functions that permits for creating an instance of that class by `$this->load->library()` function.
It is not written in object-oriented format.	It is written in an object-oriented format.
It can be called in the same manner you call PHP functions.	You must create an object of the class to call library functions by using the `$this->library_name->method()`.
All built-in helper file names are suffixed with a word `_helper` (ex: `email_helper.php`).	All built-in library files do not have a specific suffix.

What is routing in CodeIgniter?

- Routing is a technique used in CodeIgniter, by which you can define your URLs based on the requirement instead of using the predefined URLs. So, whenever there is a request made and matches the URL pattern defined by us, it will automatically direct to the specified controller and function.
- A URL string and its corresponding controller class or method are in a one-to-one relationship here. The URI segments usually follow this pattern: `example.com/class/function/id/`. All routing rules are defined in the `application/config/routes.php` file of CodeIgniter.

What are drivers in CodeIgniter?

- A driver is a type of library that has a parent class and multiple child classes. These child classes can access their parent class, but they can't access their siblings.
- Drivers can be found in the system/libraries folder.
- There are three steps for creating a driver:
  - Making file structure
  - Making driver list
  - Making driver(s)

How to link images from a view in CodeIgniter?

In Codeigniter, you can link `images/CSS/JavaScript` from a view by using the absolute path to the resources required with respect to the root folder as given below:

```
/css/styles.css
/js/query.php
/img/news/566.gpg
```

Why CodeIgniter is called a loosely based MVC framework?

Codeigniter is called a loosely based MVC framework because it does not need to obey a strict MVC pattern during application creation. It is not important to create a model, we can use only view and controllers for creating an application. In addition, one can modify CodeIgniter to utilize HMVC(Hierarchical Model View Controller) as well.

What is a helper in CodeIgniter?

- Helpers are the group of functions that are useful in assisting the user to perform specific tasks.
- There are three types of helper files. They are:
  - **URL helpers**: Used for creating the links.
  - **Text helpers**: Used for the formatting of text.
  - **Cookies helpers**: Used to read and manage cookies.

Explain CodeIgniter Architecture.

- CodeIgniter is mainly designed to deliver high performance in less time within a good environment. For achieving this, each developing process is designed in a simplified manner.
- From the technical point of view, it is dynamically instantiated (libraries are loaded only on request which makes it light-weighted), has loose coupling (components depend very less on each other), and component singularity (each class and its functions are focused only on their purpose).
- **Data flow in CodeIgniter**: Below image represents that whenever a request is raised from the CodeIgniter application, firstly, it will go to the index.php file.
  - index.php is the default file of CodeIgniter. This file initializes the base resources.
  - The router determines what should be done with the information.
  - If the requested cache file exists, then the information is moved directly to the browser and ignores the further processes.
  - If the page requested by the user does not exist in the caching file, the HTTP request and data submitted will be passed under security check.
  - The application controller will load the models, libraries, helpers, plugins, and scripts required according to the request.
  - A view is used for fetching the data from the application controller that will be represented to the user, and they pass the data to the caching file to the fastest access for future requests.

How to load a helper in CodeIgniter?

- You need to load the helper files for using it. Once loaded, it will be globally available to your controller and views. They can be obtained at two places in CodeIgniter. A helper file will be searched by CodeIgniter in the `application/helpers` folder and if it is not available in that folder then it will check in the `system/helpers` folder.
- Helper file can be loaded by adding the following code to the constructor of the controller or inside any function that wants to use: `$this->load->helper('file_name');`
  Write your file name at the place of file_name.
- To load URL helper we can use the code given below: `$this->load->helper('url');`
- You are allowed to auto-load a helper if your application needs that helper globally by including it in the `application/config/autoload.php` file.
- Loading multiple helpers is also possible. For doing this, specify them in an array as given below:

```
$this->load->helper(
   array('helper1', 'helper2', 'helper3')
);
```

What are the advantages of CodeIgniter?

Few advantages of using CodeIgniter is given below:

- **Built-in libraries**: It comes with various types of default helpers for multiple things including strings, arrays, cookies, directories, file handling, and forms among others.

- **Data abstraction**: You can make use of the CodeIgniter database abstraction layer for creating, adding, deleting, and replacing statements in a hassle-free manner. This framework allows you to manage multiple connections using a single application.
- **Active Developer Community**: Bigger the community, the better the help you get. Newly graduated developers look forward to the framework's forum to get their doubts solved and learn about new things in the process. With so many people actively participating in it from around the world, your doubts will be solved within few hours. And due to the same reason, CodeIgniter documentation is 10 times bigger than any other framework.
- **Collaboration with Expression Engine**: The collaboration permits developers using CodeIgniter to use libraries and everything else provided by Expression Engine and vice versa. Because of this, developers will get few benefits like better parser class, improved built-in user authentication, and easy access to modular applications.
- **Security**: The security strength modification can be done according to your client's needs. These changes are made when the system is initialized by switching off the `magic_quotes_runtime` directive irrespective of the `register_globals` directive. You don't need to remove the slashes during information retrieval from the database. You can enable encryption of cookies, where you can handle databases and escape SQL queries directly.
- **Immigration Features**: Database schema update management is easier over different fields by using the migration aspect. It is an easier process to immigrate from the server to the server in Codeigniter.
- **Easy to Use**: It is easier to use compared to other popular frameworks such as Symfony, Zend framework, and Cake PHP.

Give the list of hooks available in CodeIgniter.

The list of available hook points are given below:

- `pre_system`: It is called initially during system execution.
- `pre_controller`: It is called immediately before any of the controllers being called. Example:

```
$hook['pre_controller'] = array(
   'class'    => 'ExampleClass',
   'function' => 'Examplefunction',
   'filename' => 'ExampleClass.php',
   'filepath' => 'hooks',
   'params'   => array('mango', 'apple', 'orange')
   );
```

- `post_controller_constructor`: It is called soon after instantiating your controller, but before any occurrence of the method call.
- `post_controller`: It is called immediately after the complete execution of the controller.
- `display_override`: It overrides the `_display()` method.
- `cache_override`: It enables calling of the user-defined method instead of `_display_cache()` method which is available in the Output Library. This permits you for using your own cache display mechanism.
- `post_system`: It is called soon after the final rendered page has been submitted to the web browser, at the end of system execution when the final data has been sent to the browser.

What is Command-Line Interface(CLI)? Why we use CLI in Codeigniter?

Command-Line Interface or CLI is a text-based interface for interacting with computers through a set of commands. We can use CLI in CodeIgniter for:

- Running your cron-jobs without `wget` or `curl` usage
- Make your cron-jobs inaccessible from being loaded in the URL(Uniform Resource Locator) by checking the value returned by `is_cli()`
- Make interactive "tasks" that can do various things such as set permissions, run backups, prune cache folders, etc.
- It helps to integrate CodeIgniter with applications in other languages. For example, a random C++ script can call a command and run code in your models.

What is CSRF token in CodeIgniter? How to set CSRF token?

- CSRF(Cross-Site Request Forgery) token is a randomly generated value that gets modified with every HTTP request sent by webform.
- A CSRF attack forces a browser of the logged-on victim for sending a forged HTTP request, including the session cookie of the victim and other information related to authorization, to a web application. A CSRF token is used for setting or activating the protection in CodeIgniter.
- CSRF token is saved in the user's session when it is added in the website form. When we submit the form, the website compares both submitted tokens and saved tokens in the session. If they are the same, a request is considered valid. When the page gets loaded token value will also be changed each time. Thus it becomes difficult for the hackers to identify the current token.
- To set CSRF, you have to set the corresponding config value as true in your `application/config/config.php` file.
  Syntax : `$config['csrf_protection'] = TRUE;`
  If you use the form helper, the form_open() method will automatically insert a hidden CSRF field in your forms.

How to extend the class in CodeIgniter?

You have to create a file with the name Example.php under `application/core/` directory and declare your class with the below code:

```
Class Example extends CI_Input {
     // Write your code here
}
```

List various databases supported by the CodeIgniter framework.

Following Databases are supported by the CodeIgniter framework:

- MySQL (version 5.1+) database that uses MySQL (deprecated), mysqli, and PDO drivers
- Oracle database that uses oci8 and PDO drivers
- PostgreSQL database that uses Postgre and PDO drivers
- ODBC database that uses ODBC and PDO drivers
- SQLite database that uses SQLite version 2, SQLite3 version 3, along with PDO drivers
- MS SQL database that uses Sqlsrv (version 2005 and above), MsSQL, and PDO drivers
- Interbase/Firebird database that uses iBase and PDO drivers
- CUBRID database that uses Cubridand PDO drivers

What is the work of anchor tag in CodeIgniter?

- Anchor tag creates a standard HTML anchor link based on the URL of your local site.
- **Syntax:**

```
anchor($uri = '', $title = '', $attributes = '')
```

- Here, $uri represents a URI string, $title represents an anchor title and $attributes represents an HTML attributes. It returns an HTML hyperlink (anchor tag) of string type.

  The first parameter can have any segments you would like to append to the URL. These segments can be a string or an array.

  The second parameter is the text that will be displayed with a link. The URL will be used in case you leave it blank.

  The third parameter can contain an attribute list you would like added to the link. The attributes can be a string or an associative array.

- **Example:**

```
echo anchor('details/local/123', 'My Details', 'title="Details title"');
// Prints: <a href="http://example.com/index.php/details/local/123"
title="Details title">My Details</a>
```

Explain CodeIgniter E-mail library. How to send an E-mail using CodeIgniter?

- Features of Email Class in CodeIgniter are given below:
  - Multiple protocols such as Mail, Sendmail, and SMTP
  - TLS and SSL Encryption for SMTP
  - CC and BCCs
  - Multiple recipients
  - Attachments
  - HTML or Plain-text email
  - Priorities
  - Word wrapping
  - BCC Batch Mode, enabling larger e-mail lists to be broken into smaller BCC batches
  - Email Debugging tools
- **Sending Email:**
  Sending an email is a simple process here. You can configure an email on the fly or set your preferences in the `app/Config/Email.php` file. A basic example for demonstrating how you might send email is given below:

```
$email = \Config\Services::email();
$email->setFrom('your@interviewbit.com', 'Your Name');
$email->setTo('someone@interviewbit.com');
$email->setCC('another@another-example.com');
$email->setBCC('them@their-example.com');
$email->setSubject('Email Test');
$email->setMessage('Testing the email class.');
$email->send();
```

How to deal with Error handling in CodeIgniter?

CodeIgniter enables you to develop error reporting into your applications by using the below-given functions. Also, it has a class dedicated to error logging that permits messages related to error and debugging to be saved as text files.

Functions related to error handling are:

- This function will display the error message provided by the `application/errors/errorgeneral.php` template.

```
show_error('message' [, int $statuscode= 500 ] )
```

- This function shows the 404 error message supplied to it by using the `application/errors/error404.php` template.

```
show_404('page' [, 'logerror'])
```

- This function permits you to write messages onto your log files. You must provide anyone among three “levels” in the first parameter that indicates the message type (debug, error, info), with the message itself in the second parameter.

```
log_message('level', 'message')
```

Explain the default URL pattern used in CodeIgniter.

- CodeIgniter will make use of a “segment-based” approach instead of a “query string-based” approach.
- CodeIgniter framework has four main parts in the default URL pattern. First, we have the name of the server, and next, we have the name of the controller class followed by name of the controller function and function parameters at the end. Codeigniter is accessed using the URL helper. The basic URL structure is:

```
http://servername/controllerName/controllerFunction/parameter1/parameter2/.../parametern
```

- **Example:**

```
interviewbit.com/user/edit/suresh
```

How you can add or load a model in CodeIgniter?

- In CodeIgniter, models are loaded as well as called inside your controller methods. For loading a model, you must use the below-given method:

```
$this->load->model('name_of_the_model');
```

- Include the relative path from the directory of your model, if your model is placed inside a sub-directory. Consider an example, you have a model which is placed at `application/models/blog/AllPosts.php` you can load it by using: `$this->load->model('blog/AllPosts');`
- You can access the methods provided by the model, once the model gets loaded by using an object which has the same name as your controller:

```
class MyBlogController extends CI_Controller
{
   public function MyblogModel()
   {
           $this->load->model('blog_model');
           $data['que'] = $this->blog_model->get_last_five_entries();
           $this->load->view('blog_model', $data);
   }
}
```

How to pass an array from the controller to view in CodeIgniter?

A view is a webpage that shows each element of the user interface. It cannot be called directly, you need to load the views via the controller. You can pass an array from the controller to view in CodeIgniter using below given steps:

- **Create a view:**

Create a new text file and name it `ciblogview.php`. Save the created file in the `application/views/` directory. Open the `ciblogview.php` file and add the below-given code to it:

```
<html>
<head>
   <title>Blog</title>
</head>
<body>
   <h1>Welcome to Blog in CodeIgniter</h1>
```

```
</body>
</html>
```

- **Load the view:**

  Loading a view is executed using the following syntax: `$this->load->view('name');`

  Where 'name' represents the name of the view.

  The below code creates a controller named `Blog.php`. This controller has the method for loading the view.

```
<?php
class Blog extends CI_Controller
{
   public function index()
   {
       $this->load->view('ciblogview');
   }
}
?>
```

- **Passing an array from the controller to view:**

  You are allowed to paste the below-given controller code within your controller file or put it in the controller object.

```
$data['mega_header'][] = (object) array('title' => 'image portfolio' , 'img'
=> 'https://complete_image_path' );
$this->load->view('multiple_array', $data);
```

  Arrays are displayed as a brick['…'] and objects as an arrow(->). You are allowed to access an array with the brick['…'] and object using the arrow (->). Therefore, add the below-given code in the view file:

```
<?php
   if (isset($mega_header)){
       foreach ($mega_header as $key) {
           ?>
           <div class="header_item">
               <img alt="<?php echo($key['title']); ?>" src="<?php echo($key-
>img); ?>"/>
           </div>
           <?php
       }
   }
?>
```

- **Add dynamic data to views:**

  Usually, data transfer from the controller to view is done through an array or an object. The array or the object is passed as the second parameter of the view load method similar to the below-given method:

```
$data = array(
   'title' => 'TitleValue',
   'heading' => 'HeadingValue'
);
$this->load->view('ciblogview', $data);
```

The controller will look like this:

```
<?php
   class Blog extends CI_Controller {
   public function index()
   {
       $data['title'] = "TitleValue";
       $data['heading'] = "HeadingValue";
       $this->load->view('ciblogview', $data);
   }
}
?>
```

The view file will look like this:

```
<html>
<head>
   <title><?php echo $title;?></title>
</head>
<body>
   <h1><?php echo $heading;?></h1>
</body>
</html>
```

Explain how to prevent CodeIgniter from CSRF(Cross Site Request Forgery).

There are many ways to protect CodeIgniter from CSRF, one method of doing this is to use a hidden field in every form on the website. This hidden field is considered as CSRF token, it is a random value that changes with each HTTP request sent. After gets inserted into the website forms, it will be saved in the user's session as well. So, when the user submits the form, the website checks whether it is the same as the one that was saved in the session. If it is the same then, the request is authorized.

What are the sessions in CodeIgniter? How to handle sessions in CodeIgniter?

In CodeIgniter, you are allowed to maintain a user's "state" by Session class and keep an eye on their activity while they browse your website.

- **Loading a session in CodeIgniter:**
  For using session, your controller should be loaded with your Session class by using the `$this->load->library('session');`.
  Once the Session class is loaded, the Session library object can be obtained using `$this->session`.
- **Read session data in CodeIgniter:**
  `$this->session->userdata();` method of Session class is used to read or obtain session data in CodeIgniter.
  Usage: `$this->session->userdata('name_of_user');`
  Also, the below-given method of the Session class can be used to read session data.
  Usage: `$this->session->key_item`
  Where an item represents the key name you want to access.
- **Create a session in CodeIgniter:**
  The set_userdata() method that belongs to the Session class is useful in creating a session in CodeIgniter. This method uses an associative array that has the data you want to include in the session.
  Adding session data:
  Example:

```
$sessiondata = array(
   'name_of_user'  => 'lekha',
```

```
    'email'     => 'lekha@interviewbit.com',
    'log_state' => TRUE
);
$this->session->set_userdata($sessiondata);
```

If you want to add a single user data at a time, `set_userdata()` supports this syntax:

```
$this->session->set_userdata('demo_username', 'demo_value');
```

- **Remove session data in CodeIgniter:**
The unset_userdata() method that belongs to the Session class is useful for removing session data in CodeIgniter. Usage examples are given below:
Unset particular key:

```
$this->session->unset_userdata('name_of_user');
```

Unset an array of item keys:

```
$arr_items = array('name_of_user', 'email');
$this->session->unset_userdata($arr_items);
```

Explain CodeIgniter folder structure.

The CodeIgniter folder structure is given below:

- **application:** This directory will have your application logic. All of your application codes will be held in this directory. Internal subdirectories in the CodeIgniter directory structure are given below:
  - cache – It stores cached files.
  - config – It keeps configuration files.
  - controller – All application controllers are defined under this controller.
  - core – It consists of custom core classes that extend system files. For example, if you create a base controller that other controllers should extend, then you should place it under this directory.
  - helpers – This directory will be used for user-defined helper functions.
  - hooks – It is used for custom hooks in the CodeIgniter folder structure.
  - language – It is used to store language files for applications that use multiple languages.
  - libraries – It is used to store custom-created libraries.
  - logs – Application log files are placed in this directory.
  - models - All application models must be defined under this directory.
  - third_party – This is used for custom many packages that are created by you or other developers.
  - views – application views will be stored in this directory.
- **system:** It consists of the framework core files. It is not advised to make any modifications in this directory or put your own application code into this directory. System subdirectories in CodeIgniter are given below:
  - core – This is considered to be the heart of the CodeIgniter Framework. All of the core files that construct the framework are located here. If you would like to extend the core file functionality, then you must
  - create a custom core file in the application directory. After this, you are allowed to override or add new behavior that you wish. You should never make any changes directly in this directory.
  - database – It stores the files such as database drivers, cache, and other files that are needed for database operations.
  - fonts – This directory contains fonts and font-related information.
  - helpers – This directory consists of helper functions that come out of the box.
  - language – It contains language files that are used by the framework
  - libraries – It contains the source files for the different libraries that come along with CodeIgniter out of the box.
- **user_guide:** This directory consists of a user manual for CodeIgniter. You should not upload this directory during application deployment.
- **vendor:** This directory consists of composer packages source code. The `composer.json` and `composer.lock` are the other two files related to this directory.
- **index.php:** This is considered as the entry point into the application. It is placed inside the root directory.

What is the security parameter for XSS in CodeIgniter?

- Codeigniter has got a Cross-Site Scripting(XSS) hack prevention filter. This filter either automatically runs or you can run it based on item, to filter all data related to POST and COOKIE.
- The XSS filter will target the frequently used methods to trigger JavaScript code or other types of code that attempt to hijack cookies or do any other malicious activity. If it identifies anything suspicious or anything disallowed is encountered, then it will convert the data to character entities.
- To filter data through the XSS filter, we will make use of the xss_clean() method as given below:

```
$data = $this->security->xss_clean($data);
```

This function is used only when you are submitting data. The second Boolean parameter is optional and used to check the image files for the XSS attacks. This is very useful for file upload. If its value is true, that means the image is safer and not otherwise.

What is the default controller in CodeIgniter?

- When the name of the file is not mentioned in the URL then the file will be specified in the default controller that is loaded by default. By default, the file name will be `welcome.php`, which is known as the first page to be seen after the installation of CodeIgniter.
- `localhost/codeigniter/` In this case, the `welcome.php` will be generally loaded as the file name is not mentioned in the provided URL. Generally, the programmers can change the default controller that is present in the `application/config/routes.php` file as per their needs.
- `$route['default_controller'] = ' ';` In the above-given syntax, the programmer has to specify the file name that he/she wants to get loaded as the default one.

List all the auto-loadable resources available in CodeIgniter.

- The below-given items can be automatically loaded in CodeIgniter:
  - Classes obtained in the directory named `libraries/`
  - Custom config files obtained in the directory named `config/`
  - Helper files obtained in the directory named `helpers/`
  - Models obtained in the directory named `models/`
  - Language files obtained in the directory named `system/language/`
- For resource autoloading, you should open the file `application/config/autoload.php` and include the item that you want to be get loaded into the array of autoloads. In the file related to each type of item, you can find instructions.

What do you mean by the controller in CodeIgniter?

- The mediator present between the model and the view for processing the HTTP request and is used for generating a web page is called a controller. It is considered as the center of each HTTP request that exists on the web application of the user.
- Consider the following URL in this reference: `projectName/index.php/welcome/`
  In this URL, the CodeIgniter is trying to find the `welcome.php` file and the Welcome class.
- Controller syntax is given below:

```
class ControllerName extends CI_Controller
{
   public function __construct()
   {
       parent::__construct();
   }
   public function MethodName()
   {
   }
}
```

CRUD operation in CodeIgniter

https://phpgurukul.com/crud-operation-in-codeigniter/

In this tutorial we will learn about CRUD operation in CodeIgniter. CRUD Stands for create, read, update and delete record in the database. SQL table tblusers structure used in this CRUD Operation.

```sql
CREATE TABLE `tblusers` (
  `id` int(11) NOT NULL,
  `FirstName` varchar(150) NOT NULL,
  `LastName` varchar(150) NOT NULL,
  `EmailId` varchar(120) NOT NULL,
  `ContactNumber` char(11) NOT NULL,
  `Address` varchar(255) NOT NULL,
  `PostingDate` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
ALTER TABLE `tblusers` ADD PRIMARY KEY (`id`);
ALTER TABLE `tblusers`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
COMMIT;
```

1.Create a database then configure your DB credential in application/config/database.php

```php
$active_group = 'default';
$query_builder = TRUE;
$db['default'] = array(
    'dsn'   => '',
    'hostname' => 'localhost', // your hostname
    'username' => 'root', // your DB username
    'password' => '',     // DB password
    'database' => 'cicrud', // your database name
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

2. Load libraries and helpers in application/config/autoload.php that used in this CRUD operation.

```php
$autoload['libraries'] = array('form_validation','session','database');
$autoload['helper'] = array('url','html','form');
```

- Database library used for database related queries
- Session library used for display success/error message using flash data.
- form_validation library used for form validation rules.
- Html Helper for used for link_tag
- form helper used for form
- URL helper used for base_url.

For Data insertion (Create Operation)

Create a view for data insertion (insert.php) inside application/views

```
<div class="container">
<div class="row">
<div class="col-md-12">
<h3>Insert Record |  CRUD Operations using CodeIgniter</h3>
<hr />
</div>
</div>
<?php echo form_open('',['name'=>'insertdata','autocomplete'=>'off']);?>
<div class="row">
<div class="col-md-4"><b>First Name</b>
<?php echo form_input(['name'=>'firstname','class'=>'form-control','value'=>set_value('firstnam
e')]);?>
<?php echo form_error('firstname',"<div style='color:red'>","</div>");?>
</div>
<div class="col-md-4"><b>Last Name</b>
<?php echo form_input(['name'=>'lastname','class'=>'form-control','value'=>set_value('lastname')]);?>
<?php echo form_error('lastname',"<div style='color:red'>","</div>");?>
</div>
</div>
<div class="row">
<div class="col-md-4"><b>Email id</b>
<?php echo form_input(['name'=>'emailid','class'=>'form-control','value'=>set_value('emailid')]);?>
<?php echo form_error('emailid',"<div style='color:red'>","</div>");?>
</div>
<div class="col-md-4"><b>Contactno</b>
<?php echo form_input(['name'=>'contactno','class'=>'form-control','value'=>set_value('contactn
o')]);?>
<?php echo form_error('contactno',"<div style='color:red'>","</div>");?>
</div>
```

```
</div>
<div class="row">
<div class="col-md-8"><b>Address</b>
<?php echo form_textarea(['name'=>'address','class'=>'form-control','value'=>set_value('address')]);?
>
<?php echo form_error('address',"<div style='color:red'>","</div>");?>
</div>
</div>
<div class="row" style="margin-top:1%">
<div class="col-md-8">
<?php echo form_submit(['name'=>'insert','value'=>'Submit']);?>
</div>
</div>
<?php echo form_close();?>
</div>
</div>
```

Create a controller for data insertion (Insert.php) inside application/controller. Set the validation rules in this controller.

```php
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
class Insert extends CI_Controller {
// For data insertion
public function index(){
//Setting validation rules
$this->form_validation->set_rules('firstname','First Name','required|alpha');
$this->form_validation->set_rules('lastname','Last Name','required|alpha');
$this->form_validation->set_rules('emailid','Email id','required|valid_email');
$this->form_validation->set_rules('contactno','Contact Number','required|numeric|exact_length[10]');
$this->form_validation->set_rules('address','Address','required');
if($this->form_validation->run()){
$fname=$this->input->post('firstname');
$lname=$this->input->post('lastname');
$email=$this->input->post('emailid');
$cntno=$this->input->post('contactno');
$adrss=$this->input->post('address');
//loading model
$this->load->model('Insert_Model');
$this->Insert_Model->insertdata($fname,$lname,$email,$cntno,$adrss);
$this->load->view('insert');
} else {
$this->load->view('insert');
}
}
}
```

Create a model (Insert_Model.php) inside application/model.

```php
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
class Insert_Model extends CI_Model {
public function insertdata($fname,$lname,$email,$cntno,$adrss){
$data=array(
            'FirstName'=>$fname,
            'LastName'=>$lname,
            'EmailId'=>$email,
            'ContactNumber'=>$cntno,
            'Address'=>$adrss
        );
$sql_query=$this->db->insert('tblusers',$data);
if($sql_query){
$this->session->set_flashdata('success', 'Registration successful');
        redirect('read');
    }
    else{
        $this->session->set_flashdata('error', 'Somthing went worng. Error!!');
        redirect('read');
    }
    }}
```

For Data Fetching (Read Operation)

Create a model (Read_Model.php inside application/models) for reading data from database using Active Records .

```php
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
class Read_Model extends CI_Model{
public function getdata(){
$query=$this->db->select('FirstName,LastName,EmailId,ContactNumber,Address,PostingDate,id')
        ->get('tblusers');
         return $query->result();
    }
}
```

Now create a controller (Read.php) inside application/controller.

```php
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
class Read extends CI_Controller{
// for all records
public function index(){
//loading model
$this->load->model('Read_Model');
$results=$this->Read_Model->getdata();
// Passing values to view
$this->load->view('read',['result'=>$results]);
}
}
```

Create a view(read.php inside application/views) for showing data.

```php
<div class="container">
<div class="row">
<div class="col-md-12">
<h3>CRUD Operations using CodeIgniter</h3> <hr />
<!--- Success Message --->
<?php if ($this->session->flashdata('success')) { ?>
<p style="font-size: 20px; color:green"><?php echo $this->session->flashdata('success'); ?></p>
<?php }?>
<!---- Error Message ---->
<?php if ($this->session->flashdata('error')) { ?>
<p style="font-size: 20px; color:red"><?php echo $this->session->flashdata('error'); ?></p>
 <?php } ?>
<a href="<?php echo site_url('insert'); ?>">
<button class="btn btn-primary"> Insert Record</button></a>
<div class="table-responsive">
<table id="mytable" class="table table-bordred table-striped">
<thead>
<th>#</th>
<th>First Name</th>
<th>Last Name</th>
<th>Email</th>
<th>Contact</th>
<th>Address</th>
<th>Posting Date</th>
<th>Edit</th>
<th>Delete</th>
</thead>
<tbody>
```

```php
<?php
$cnt=1;
foreach($result as $row)
{
?>
    <tr>
    <td><?php echo htmlentities($cnt);?></td>
    <td><?php echo htmlentities($row->FirstName);?></td>
    <td><?php echo htmlentities($row->LastName);?></td>
    <td><?php echo htmlentities($row->EmailId);?></td>
    <td><?php echo htmlentities($row->ContactNumber);?></td>
    <td><?php echo htmlentities($row->Address);?></td>
    <td><?php echo htmlentities($row->PostingDate);?></td>
    <td>
<?php
//for passing row id to controller
 echo  anchor("Read/getdetails/{$row->id}",' ','class="btn btn-primary btn-xs glyphicon glyphicon-pencil"')?>
</td>
<td>
<?php
//for passing row id to controller
 echo anchor("Delete/index/{$row->id}",' ','class="glyphicon glyphicon-trash btn-danger btn-xs"')?>
</td>
</tr>
<?php
// for serial number increment
$cnt++;
} ?>
</tbody>
</table>
</div>
</div>
</div>
</div>
```

For Data Updation (Update Operation)

In previous operation we fetched all data from database. Now whenever user will click on the edit button row id passed to controller on the basis of that row id we will fetch and update the data. First we will fetch the particular row data then update . We already created a controller Read.php now add a function with parameter “getdetails($uid)” in this controller for fetch particular record.

```php
// for particular recod
public function getdetails($uid)
{
//loading model
$this->load->model('Read_Model');
$reslt=$this->Read_Model->getuserdetail($uid);
// Passing Values to update view
$this->load->view('update',['row'=>$reslt]);
}
```

Now we will add a new function “getuserdetail($uid)” in Read_Model.php for fetching particular row data.

```php
public function getuserdetail($uid){
        $ret=$this->db->select('FirstName,LastName,EmailId,ContactNumber,Address,PostingDate,id')
                      ->where('id',$uid)
                      ->get('tblusers');
                return $ret->row();
    }
```

Create a view for data updation(update.php) inside application/views.

```
<div class="container">
<div class="row">
<div class="col-md-12">
<h3>Update Record |  CRUD Operations using CodeIgniter</h3>
<hr />
</div>
</div>
<!--- Success Message --->
<?php if ($this->session->flashdata('success')) { ?>
<p style="font-size: 18px; color:green"><?php echo $this->session->flashdata('success'); ?></p>
<?php }?>
<!---- Error Message ---->
<?php if ($this->session->flashdata('error')) { ?>
<p style="font-size: 18px; color:red"><?php echo $this->session->flashdata('error'); ?></p>
 <?php } ?>
<?php echo form_open('Insert/updatedetails',['name'=>'insertdata','autocomplete'=>'off']);?>
<?php echo form_hidden('userid',$row->id);?>
<div class="row">
<div class="col-md-4"><b>First Name</b>
<?php echo form_input(['name'=>'firstname','class'=>'form-control','value'=>set_value('firstname',$ro
w->FirstName)]);?>
<?php echo form_error('firstname',"<div style='color:red'>","</div>");?>
</div>
<div class="col-md-4"><b>Last Name</b>
<?php echo form_input(['name'=>'lastname','class'=>'form-control','value'=>set_value('lastname',$row-
>LastName)]);?>
<?php echo form_error('lastname',"<div style='color:red'>","</div>");?>
</div>
</div>

<div class="row">
<div class="col-md-4"><b>Email id</b>
<?php echo form_input(['name'=>'emailid','class'=>'form-control','value'=>set_value('emailid',$row->E
mailId)]);?>
<?php echo form_error('emailid',"<div style='color:red'>","</div>");?>
</div>
<div class="col-md-4"><b>Contactno</b>
<?php echo form_input(['name'=>'contactno','class'=>'form-control','value'=>set_value('contactno',$ro
w->ContactNumber)]);?>
<?php echo form_error('contactno',"<div style='color:red'>","</div>");?>
</div>
</div>
<div class="row">
<div class="col-md-8"><b>Address</b>
<?php echo form_textarea(['name'=>'address','class'=>'form-control','value'=>set_value('address',$row
->Address)]);?>
<?php echo form_error('address',"<div style='color:red'>","</div>");?>
</div>
</div>
<div class="row" style="margin-top:1%">
<div class="col-md-8">
<?php echo form_submit(['name'=>'insert','value'=>'Update']);?>
</div>
</div>
<?php echo form_close();?>
</div>
</div>
```

Now add a function “updatedetails” in Insert.php for data updation and set validation rules in this controller.

```php
// For data updation
public function updatedetails(){
$this->form_validation->set_rules('firstname','First Name','required|alpha');
$this->form_validation->set_rules('lastname','Last Name','required|alpha');
$this->form_validation->set_rules('emailid','Email id','required|valid_email');
$this->form_validation->set_rules('contactno','Contact Number','required|numeric|exact_length[10]');
$this->form_validation->set_rules('address','Address','required');
if($this->form_validation->run()){
$fname=$this->input->post('firstname');
$lname=$this->input->post('lastname');
$email=$this->input->post('emailid');
$cntno=$this->input->post('contactno');
$adrss=$this->input->post('address');
$usid=$this->input->post('userid');
$this->load->model('Insert_Model');
$this->Insert_Model->updatedetails($fname,$lname,$email,$cntno,$adrss,$usid);
} else {
$this->session->set_flashdata('error', 'Somthing went worng. Try again with valid details !!');
        redirect('read');
}
}
```

After this add a new function “updatedetails” in Insert_Model.php for data updation.

```php
public function updatedetails($fname,$lname,$email,$cntno,$adrss,$usid){
$data=array(
            'FirstName'=>$fname,
            'LastName'=>$lname,
            'EmailId'=>$email,
            'ContactNumber'=>$cntno,
            'Address'=>$adrss
        );
$sql_query=$this->db->where('id', $usid)
                ->update('tblusers', $data);
           if($sql_query){
$this->session->set_flashdata('success', 'Record updated successful');
        redirect('read');
    }
    else{
        $this->session->set_flashdata('error', 'Somthing went worng. Error!!');
        redirect('read');
    }
```

For Data Deletion (Delete Operation)

Create a controller (Delete.php) inside application/contollers.

```php
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
class Delete extends CI_Controller{
public function index($uid)
{
$this->load->model('Delete_Model');
$this->Delete_Model->deleterow($uid);
$this->load->view('read');
}
}
```

Now create a model for data deletion(Delete_Model.php) inside application/model.

```php
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
class Delete_Model extends CI_Model {
public function deleterow($uid){
$sql_query=$this->db->where('id', $uid)
                ->delete('tblusers');
            if($sql_query){
$this->session->set_flashdata('success', 'Record delete successfully');
        redirect('read');
    }
    else{
        $this->session->set_flashdata('error', 'Somthing went worng. Error!!');
        redirect('read');
    }
}
}
```

Chatgpt Crud

This example will cover all CRUD operations (Create, Read, Update, Delete) for managing products.

1. **Setup Database**:

   First, let's create a MySQL database named `product_management` and a table named `products` with the following structure:

```sql
CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    description TEXT
);
```

2. **Create CodeIgniter Project**:

   Set up a new CodeIgniter 4 project if you haven't already. You can do this using Composer:

```bash
composer create-project codeigniter4/appstarter product_management
```

3. **Create Model (`ProductModel.php`):**

   Create a model file `ProductModel.php` inside the `app/Models` directory. This model will handle interactions with the `products` table:

```php
<?php

namespace App\Models;

use CodeIgniter\Model;

class ProductModel extends Model
{
    protected $table = 'products';
    protected $primaryKey = 'id';
    protected $allowedFields = ['name', 'price', 'description'];
}
```

4. **Create Controller (`Products.php`):**

   Create a controller file `Products.php` inside the `app/Controllers` directory. This controller will handle CRUD operations for products:

```php
<?php

namespace App\Controllers;

use App\Models\ProductModel;
use CodeIgniter\Controller;

class Products extends Controller
{
    public function index()
    {
        $productModel = new ProductModel();
        $data['products'] = $productModel->findAll();
        return view('products/index', $data);
    }
```

```php
public function create()
{
    if ($this->request->getMethod() === 'post') {
        $productModel = new ProductModel();
        $data = [
            'name' => $this->request->getPost('name'),
            'price' => $this->request->getPost('price'),
            'description' => $this->request->getPost('description'),
        ];
        $productModel->insert($data);
        return redirect()->to('/products');
    }
    return view('products/create');
}

public function edit($id)
{
    $productModel = new ProductModel();
    $data['product'] = $productModel->find($id);
    return view('products/edit', $data);
}
```

```php
    public function update($id)
    {
        if ($this->request->getMethod() === 'post') {
            $productModel = new ProductModel();
            $data = [
                'name' => $this->request->getPost('name'),
                'price' => $this->request->getPost('price'),
                'description' => $this->request->getPost('description'),
            ];
            $productModel->update($id, $data);
            return redirect()->to('/products');
        }
    }

    public function delete($id)
    {
        $productModel = new ProductModel();
        $productModel->delete($id);
        return redirect()->to('/p cts');
    }
```

5. **Create Views**:
   - Create a directory named `products` inside the `app/Views` directory.
   - Create `index.php`, `create.php`, and `edit.php` views inside the `app/Views/products` directory.

`index.php`:

php Copy code

```php
<h1>Products</h1>
<a href="/products/create">Add Product</a>
<ul>
    <?php foreach ($products as $product): ?>
        <li>
            <strong><?php echo $product['name']; ?></strong> - $<?php echo $product[
            <a href="/products/edit/<?php echo $product['id']; ?>">Edit</a>
            <a href="/products/delete/<?php echo $product['id']; ?>" onclick="return
        </li>
    <?php endforeach; ?>
</ul>
```

```
ct['name']; ?></strong> - $<?php echo $product['price']; ?>
?php echo $product['id']; ?>">Edit</a>
/<?php echo $product['id']; ?>" onclick="return confirm('Are you sure?')">Delete</a>
```

`create.php`:

```php
<h1>Add Product</h1>
<form action="/products/create" method="post">
    <input type="text" name="name" placeholder="Product Name">
    <input type="text" name="price" placeholder="Price">
    <textarea name="description" placeholder="Description"></textarea>
    <button type="submit">Add Product</button>
</form>
```

`edit.php`:

```php
<h1>Edit Product</h1>
<form action="/products/update/<?php echo $product['id']; ?>" method="post">
    <input type="text" name="name" value="<?php echo $product['name']; ?>">
    <input type="text" name="price" value="<?php echo $product['price']; ?>">
    <textarea name="description"><?php echo $product['description']; ?></textarea>
    <button type="submit">Update Product</button>
</form>
```

6. **Routes:**

   Open `app/Config/Routes.php` and define routes for the `Products` controller:

```php
$routes->get('products', 'Products::index');
$routes->get('products/create', 'Products::create');
$routes->post('products/create', 'Products::create');
$routes->get('products/edit/(:num)', 'Products::edit/$1');
$routes->post('products/update/(:num)', 'Products::update/$1');
$routes->get('products/delete/(:num)', 'Products::delete/$1');
```

7. **Testing**:

   You can now test the CRUD operations by accessing the following URLs:

   - `/products`: View all products
   - `/products/create`: Add a new product
   - `/products/edit/{id}`: Edit an existing product
   - `/products/delete/{id}`: Delete an existing product

PHP Tech Life Channel CRUD

https://www.youtube.com/watch?v=v6LzNJZ_KLU

"Simple Codeigniter CRUD application in Hindi (Create, Read, Update, Delete)" is a video tutorial based on Codeigniter 3, MySql and Bootstrap 4 in this video you will learn following concepts.

- How to configure a Codeigniter framework using config.php

- How to configure a database connection using database.php file

- How to autoload files libraries, helpers, models, etc. using autoload.php file

- How to create a Model in Codeigniter

- How to create a Controller in Codeigniter

- How to create a Model in Codeigniter

- How to use Active Query Builder in Codeigniter

- How to insert a record in a database.

- How to select a record in a database

- How to edit/update a record in a database

- How to delete a record in a database

- How to use session in Codeigniter

- How to use bootstrap in Codeigniter views.

You can follow the instructions given below described in the video:- CRUD Application C :- Create R :- Read U :- Update D :- Delete We will follow the following steps to create a crud application in Codeigniter framework.

1) Download and Extract the Codeigniter framework in the htdocs folder.

2) Configure Codeigniter using config.php and database.php and autoload libraries like session, database, form_validation, etc.

3) Create a users table in the database using phpMyAdmin.

4) Create a controller called User in the controllers folder.

6) Create views called list, create, edit in views folder.

7) Create a model called User Model in the models folder.

Add User:-

8) To show Add User form, create a method called "create" in User controller.

9) Load create view in "create" method. 10) Write a "create" method in the User model to save a User in the database.

11) Apply validation and save form values in database and show success messages.

List Users:-

12) Create a method called "index" & Load "list" view in User controller to show all users using User_model.

13) Write an "All" method in the User model for fetching all the user records.

14) Write a loop to list all users in "list" view, also create an EDIT and DELETE button with links.

Edit User:-

15) Create an edit method in User controller to edit a User

16) Write a "getUser" method in the User model to fetch a single user row.

17) Load edit user view to edit a user with prepopulate user data.

18) Write a "updateUser" method in the User model to update a user record.

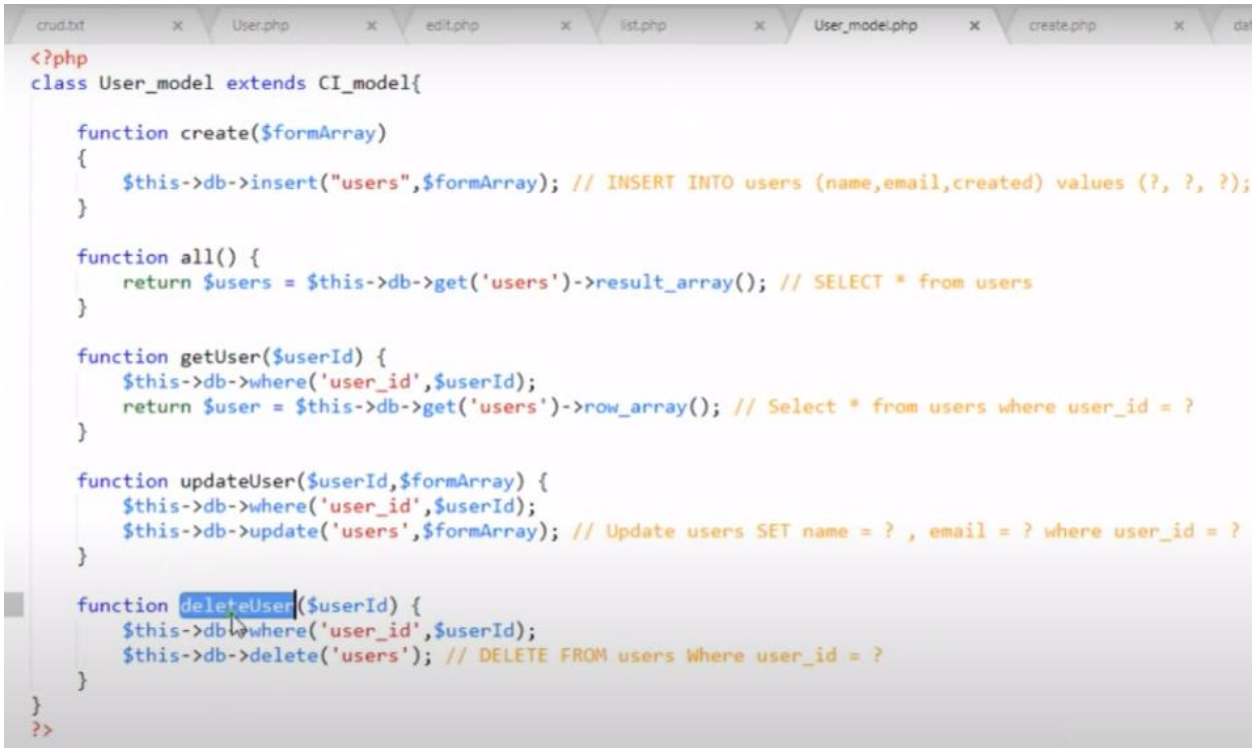
19) Apply validation & update form values in database and show success notification.

Delete :-

20) Create a delete method in the User controller.

21) Write a "deleteUser" method in the User model to delete a user record from the database.

22) Check if a record found in DB, if found then delete and redirect to list page with success message, else redirect with an error message.



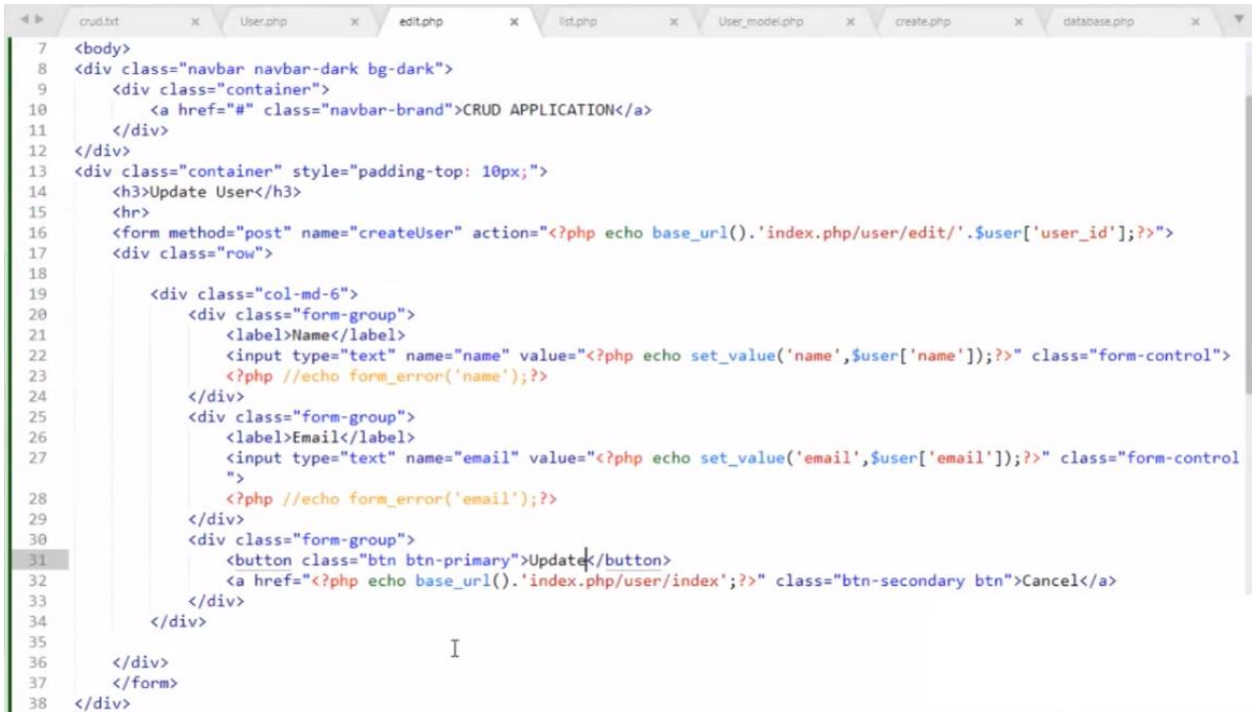
User controller

```php
<?php
class User extends CI_controller{

    function index() {
        $this->load->model('User_model');
        $users = $this->User_model->all();
        $data = array();
        $data['users'] = $users;
        $this->load->view('list',$data);
    }

    function create() {
        $this->load->model('User_model');
        $this->form_validation->set_rules('name','Name','required');
        $this->form_validation->set_rules('email','Email','required|valid_email');

        if ($this->form_validation->run() == false) {
            $this->load->view('create');
        } else {
            // Save record to database
            $formArray = array();
            $formArray['name'] = $this->input->post('name');
            $formArray['email'] = $this->input->post('email');
            $formArray['created_at'] = date('Y-m-d');
            $this->User_model->create($formArray);
            $this->session->set_flashdata('success','Record added successfully!');
            redirect(base_url().'index.php/user/index');
        }

    }

    function edit($userId)
    {
        $this->load->model('User_model');
        $user = $this->User_model->getUser($userId);
        $data = array();
        $data['user'] = $user;

        $this->form_validation->set_rules('name','Name','required');
        $this->form_validation->set_rules('email','Email','required|valid_email');

        if ($this->form_validation->run() == false)  {
            $this->load->view('edit',$data);
        } else {
            // update user record
            $formArray = array();
            $formArray['name'] = $this->input->post('name');
            $formArray['email'] = $this->input->post('email');
            $this->User_model->updateUser($userId,$formArray);
            $this->session->set_flashdata('success','Record updated successfully');
            redirect(base_url().'index.php/user/index');
        }


    }
}
```

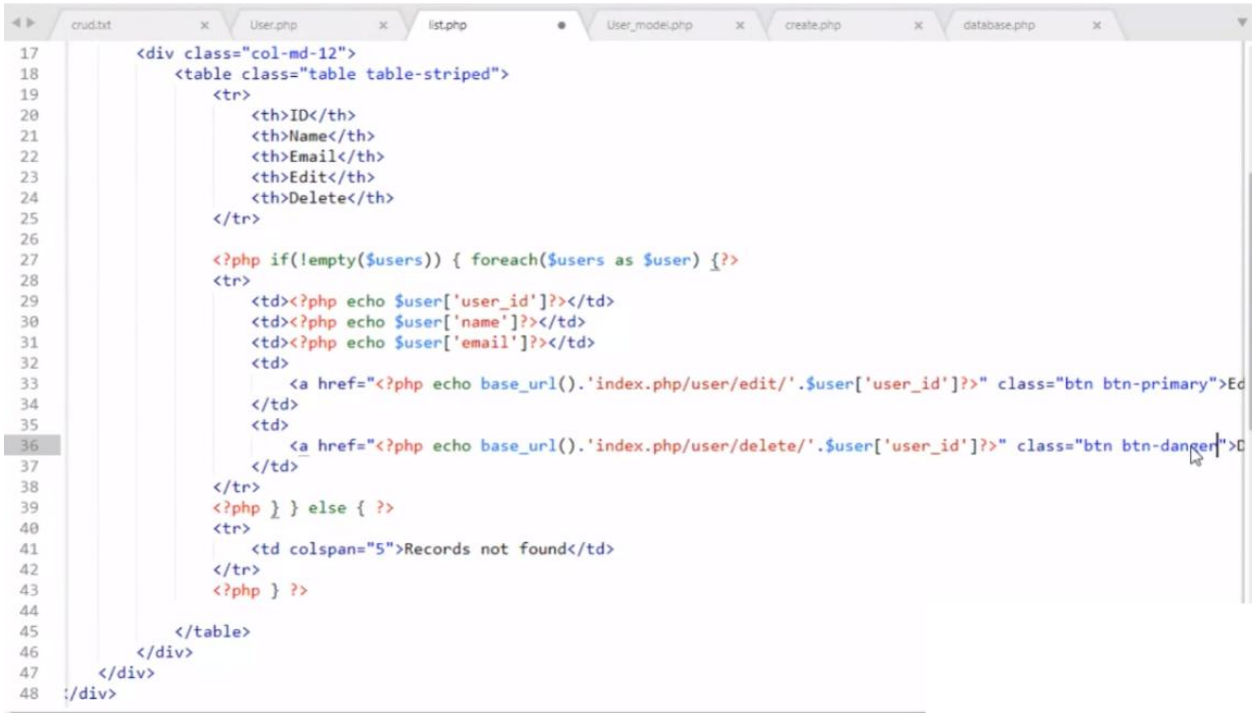
```php
    function delete($userId)
    {
        $this->load->model('User_model');
        $user = $this->User_model->getUser($userId);
        if (empty($user)) {
            $this->session->set_flashdata('failure','Record not found in database');
            redirect(base_url().'index.php/user/index');
        }

        $this->User_model->deleteUser($userId);
        $this->session->set_flashdata('succcess','Record deleted successfully');
        redirect(base_url().'index.php/user/index');

    }
}
```

crud.txt | User.php | edit.php | list.php | User_model.php | create.php | data

```php
<html>
<head>
    <title>Crud Application - View Users</title>
    <link rel="stylesheet" type="text/css" href="<?php echo base_url().'assets/css/bootstrap.min.css';?>">
</head>
<body>
<div class="navbar navbar-dark bg-dark">
    <div class="container">
        <a href="#" class="navbar-brand">CRUD APPLICATION</a>
    </div>
</div>
<div class="container" style="padding-top: 10px;">
    <div class="row">
        <div class="col-md-12">
            <?php
            $success = $this->session->userdata('success');
            if($success != "") {
            ?>
            <div class="alert alert-success"><?php echo $success;?></div>
            <?php
            }
            ?>
            <?php
            $failure = $this->session->userdata('failure');
            if($failure != "") {
            ?>
            <div class="alert alert-success"><?php echo $failure;?></div>
            <?php
            }
            ?>
        </div>
    </div>
```



```
crud.txt | User.php | edit.php | list.php | User_model.php | create.php | database.php

<body>
<div class="navbar navbar-dark bg-dark">
    <div class="container">
        <a href="#" class="navbar-brand">CRUD APPLICATION</a>
    </div>
</div>
<div class="container" style="padding-top: 10px;">
    <h3>Update User</h3>
    <hr>
    <form method="post" name="createUser" action="<?php echo base_url().'index.php/user/edit/'.$user['user_id'];?>">
    <div class="row">

        <div class="col-md-6">
            <div class="form-group">
                <label>Name</label>
                <input type="text" name="name" value="<?php echo set_value('name',$user['name']);?>" class="form-control">
                <?php //echo form_error('name');?>
            </div>
            <div class="form-group">
                <label>Email</label>
                <input type="text" name="email" value="<?php echo set_value('email',$user['email']);?>" class="form-control
                ">
                <?php //echo form_error('email');?>
            </div>
            <div class="form-group">
                <button class="btn btn-primary">Update</button>
                <a href="<?php echo base_url().'index.php/user/index';?>" class="btn-secondary btn">Cancel</a>
            </div>
        </div>

    </div>
    </form>
</div>
```



```
crud.txt | User.php | list.php | User_model.php | create.php | database.php

        <div class="col-md-12">
            <table class="table table-striped">
                <tr>
                    <th>ID</th>
                    <th>Name</th>
                    <th>Email</th>
                    <th>Edit</th>
                    <th>Delete</th>
                </tr>

                <?php if(!empty($users)) { foreach($users as $user) {?>
                <tr>
                    <td><?php echo $user['user_id']?></td>
                    <td><?php echo $user['name']?></td>
                    <td><?php echo $user['email']?></td>
                    <td>
                        <a href="<?php echo base_url().'index.php/user/edit/'.$user['user_id']?>" class="btn btn-primary">Ed
                    </td>
                    <td>
                        <a href="<?php echo base_url().'index.php/user/delete/'.$user['user_id']?>" class="btn btn-danger">D
                    </td>
                </tr>
                <?php } } else { ?>
                <tr>
                    <td colspan="5">Records not found</td>
                </tr>
                <?php } ?>

            </table>
        </div>
    </div>
</div>
```

Api in CodeIgniter

1. **Define Routes:**

   Open the `app/Config/Routes.php` file and define routes for your API endpoints.

```php
$routes->get('api/products', 'ProductController::index');
$routes->get('api/products/(:num)', 'ProductController::show/$1');
$routes->post('api/products', 'ProductController::create');
$routes->put('api/products/(:num)', 'ProductController::update/$1');
$routes->delete('api/products/(:num)', 'ProductController::delete/$1');
```

2. **Create Controller:**

   Create a controller named `ProductController.php` inside the `app/Controllers` directory.

```php
<?php

namespace App\Controllers;

use App\Models\ProductModel;
use CodeIgniter\API\ResponseTrait;
use CodeIgniter\Controller;

class ProductController extends Controller
{
    use ResponseTrait;

    public function index()
    {
        $productModel = new ProductModel();
        $products = $productModel->findAll();

        return $this->respond($products);
    }
```

```php
public function show($id)
{
    $productModel = new ProductModel();
    $product = $productModel->find($id);

    if (!$product) {
        return $this->failNotFound('Product not found');
    }

    return $this->respond($product);
}
```

```php
public function create()
{
    $rules = [
        'name' => 'required',
        'price' => 'required|numeric',
        // Add more validation rules as needed
    ];

    if (!$this->validate($rules)) {
        return $this->failValidationErrors($this->validator->getErrors());
    }

    $productModel = new ProductModel();
    $productId = $productModel->insert($this->request->getPost());

    return $this->respondCreated(['id' => $productId]);
}
```

```php
    public function update($id)
    {
        $productModel = new ProductModel();
        $product = $productModel->find($id);

        if (!$product) {
            return $this->failNotFound('Product not found');
        }

        $rules = [
            'name' => 'required',
            'price' => 'required|numeric',
            // Add more validation rules as needed
        ];

        if (!$this->validate($rules)) {
            return $this->failValidationErrors($this->validator->getErrors());
        }

        $productModel->update($id, $this->request->getJSON());

        return $this->respondUpdated(['id' => $id]);
    }
```

```php
    public function delete($id)
    {
        $productModel = new ProductModel();
        $product = $productModel->find($id);

        if (!$product) {
            return $this->failNotFound('Product not found');
        }

        $productModel->delete($id);

        return $this->respondDeleted(['id' => $id]);
    }
}
```

3. **Create Model**:

   Create a model named `**ProductModel.php**` inside the `**app/Models**` directory to interact with the `**products**` table in your database.

```php
<?php

namespace App\Models;

use CodeIgniter\Model;

class ProductModel extends Model
{
    protected $table = 'products';
    protected $primaryKey = 'id';
    protected $allowedFields = ['name', 'price'];
}
```

4. **Database Configuration**:

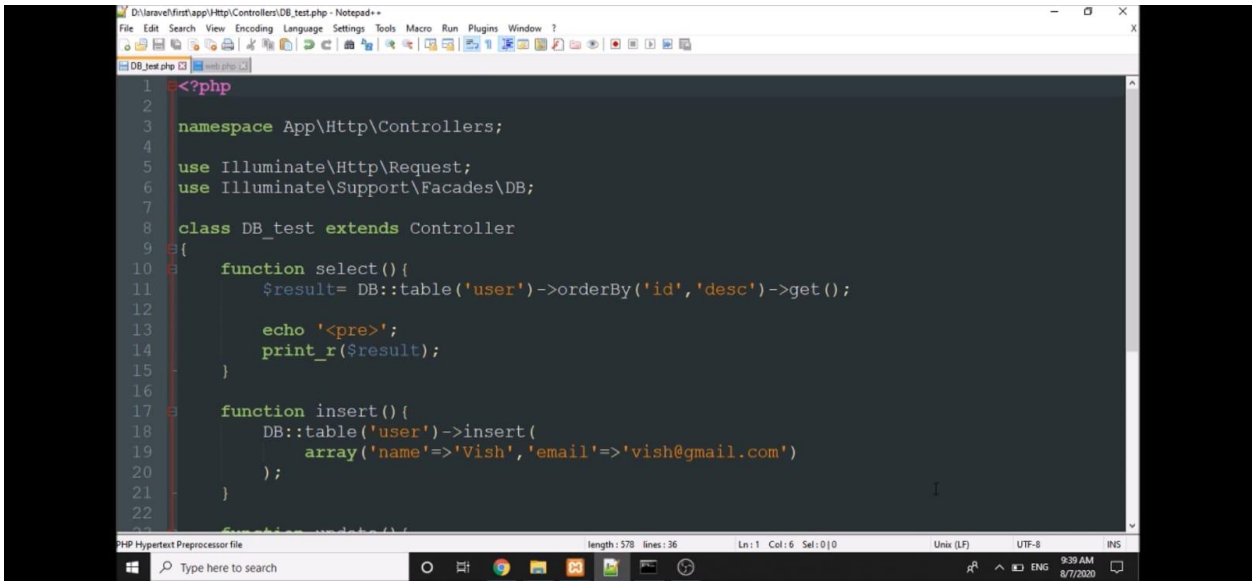
   Ensure your database connection settings are properly configured in the `**app/Config/Database.php**` file.

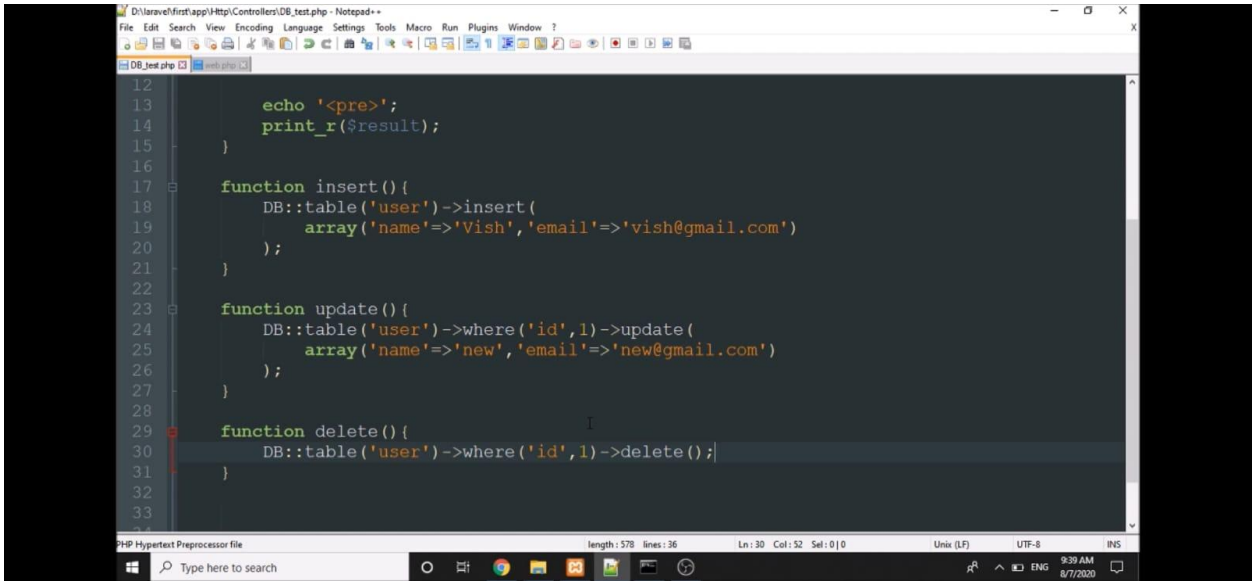
5. **Input Validation**:

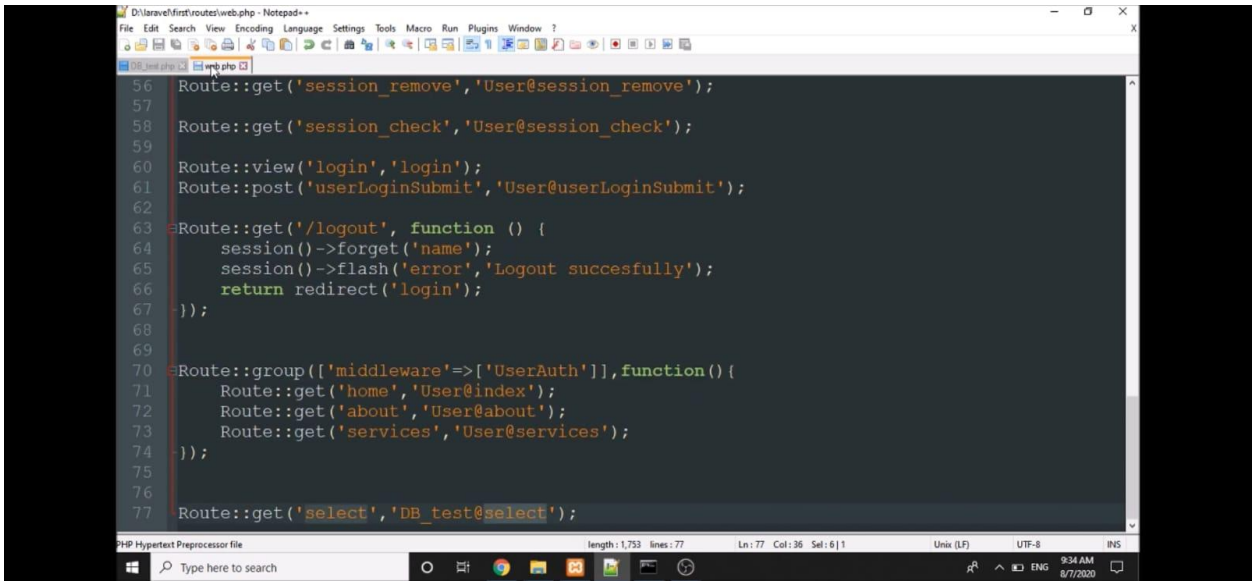
   You can use CodeIgniter's built-in validation library or manual validation methods to validate input data in your controller methods.

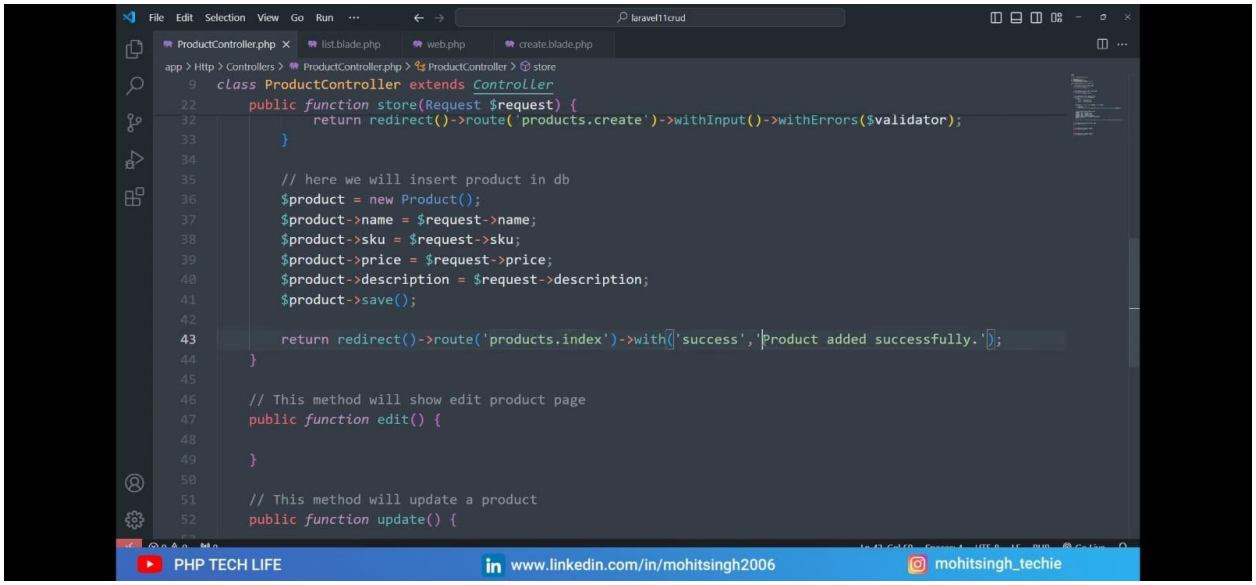
6. **Error Handling**:

   Handle errors and exceptions appropriately in your controller methods and return proper HTTP responses with error messages when necessary.


D:\laravel\first\app\Http\Controllers\DB_test.php - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
DB_test.php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
class DB_test extends Controller
{
function select(){
$result= DB::table('user')->orderBy('id','desc')->get();
echo '<pre>';
print_r($result);
}
function insert(){
DB::table('user')->insert(
array('name'=>'Vish','email'=>'vish@gmail.com')
);
}
PHP Hypertext Preprocessor file
length : 578 lines : 36
Ln : 1 Col : 6 Sel : 0 | 0
Unix (LF)
UTF-8
INS
Type here to search
ENG
9:39 AM
8/7/2020


D:\laravel\first\app\Http\Controllers\DB_test.php - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
DB_test.php
echo '<pre>';
print_r($result);
}
function insert(){
DB::table('user')->insert(
array('name'=>'Vish','email'=>'vish@gmail.com')
);
}
function update(){
DB::table('user')->where('id',1)->update(
array('name'=>'new','email'=>'new@gmail.com')
);
}
function delete(){
DB::table('user')->where('id',1)->delete();
}
PHP Hypertext Preprocessor file
length : 578 lines : 36
Ln : 30 Col : 52 Sel : 0 | 0
Unix (LF)
UTF-8
INS
Type here to search
ENG
9:39 AM
8/7/2020


D:\laravel\first\routes\web.php - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
web.php
Route::get('session_remove','User@session_remove');
Route::get('session_check','User@session_check');
Route::view('login','login');
Route::post('userLoginSubmit','User@userLoginSubmit');
Route::get('/logout', function () {
session()->forget('name');
session()->flash('error','Logout succesfully');
return redirect('login');
});
Route::group(['middleware'=>['UserAuth']],function(){
Route::get('home','User@index');
Route::get('about','User@about');
Route::get('services','User@services');
});
Route::get('select','DB_test@select');
PHP Hypertext Preprocessor file
length : 1,753 lines : 77
Ln : 77 Col : 36 Sel : 6 | 1
Unix (LF)
UTF-8
INS
Type here to search
ENG
9:34 AM
8/7/2020


File Edit Selection View Go Run
laravel11crud
ProductController.php
list.blade.php
web.php
create.blade.php
app > Http > Controllers > ProductController.php > ProductController > store
class ProductController extends Controller
public function store(Request $request) {
return redirect()->route('products.create')->withInput()->withErrors($validator);
}
// here we will insert product in db
$product = new Product();
$product->name = $request->name;
$product->sku = $request->sku;
$product->price = $request->price;
$product->description = $request->description;
$product->save();
return redirect()->route('products.index')->with('success','Product added successfully.');
}
// This method will show edit product page
public function edit() {
}
// This method will update a product
public function update() {
PHP TECH LIFE
www.linkedin.com/in/mohitsingh2006
mohitsingh_techie

What is new in Laravel 11 ?

- Minimum Requirement PHP 8.2
- Improve Folder Structure
- Default Database – SQLite
- Laravel Reverb - WebSocket Server
- New Once method
- Model casts changes
- Per-Second Rate Limiting
- Eager Load Limit
- New Artisan Commands
- Named Arguments Removed

Laravel : One To One Relationship

Students Table

id	Name	Age
1	Ram Kumar	19
2	Salman Khan	18
3	Meera Khan	19
4	Sarita Kumari	21

Contacts Table

id	Phone	City	Student_id
1	9988441122	Agra	1
2	8833554477	Mumbai	2
3	2233665544	Delhi	3
4	1122334455	Mumbai	4

1. App/Models/Student.php ⇢ App/Models/Contact.php

2.
```
public function contact(){
    return $this->hasOne(Contact::class);
}
```

Laravel : Eloquent Model Conventions

App/Models/User.php

```
class User extends Model{
    use HasUuids;    36 characters long
    use HasUlids;    26 characters long

}
```


Laravel : Eloquent Model Conventions

App/Models/User.php

```
class User extends Model{
    protected $table = 'users_data';
    protected $primaryKey = 'user_id';
    public $incrementing = false;
    protected $keyType = 'string';
    public $timestamps = false;
}
```

```
class User extends Model{
    protected $dateFormat = 'U';
    const CREATED_AT = 'creation_date';
    const UPDATED_AT = 'updated_date';
    protected $attributes = [
      'age' => 18,
      'city' => 'Goa',
    ];
    protected $connection = 'sqlite';
}
```

Laravel : One To One Relationship

Students Table

id	Name	Age
1	Ram Kumar	19
2	Salman Khan	18
3	Meera Khan	19
4	Sarita Kumari	21

PRIMARY KEY

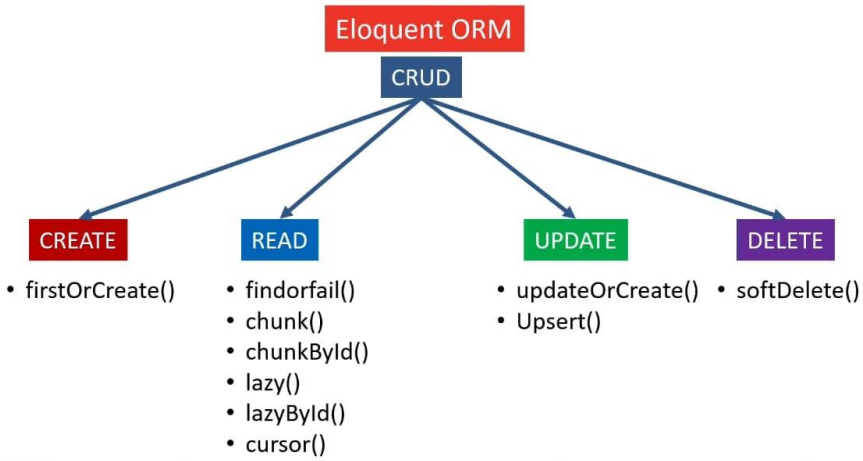
Contacts Table

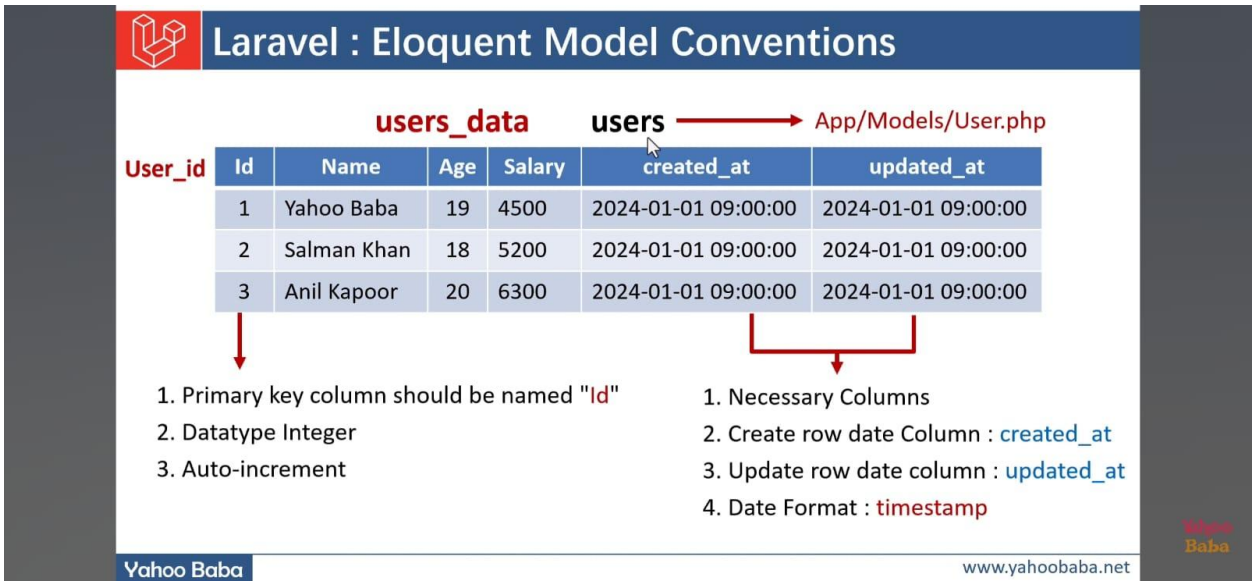
id	Phone	City	Student_id
1	9988441122	Agra	1
2	8833554477	Mumbai	2
3	2233665544	Delhi	3
4	1122334455	Mumbai	4

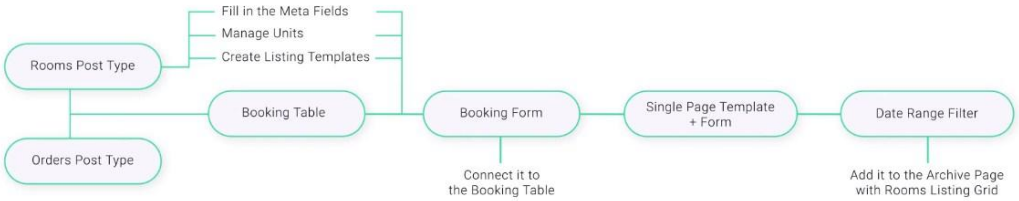
FOREIGN KEY

```
$table->foreign('Student_id')->references('id')->on('students');
```


Laravel : Eloquent ORM

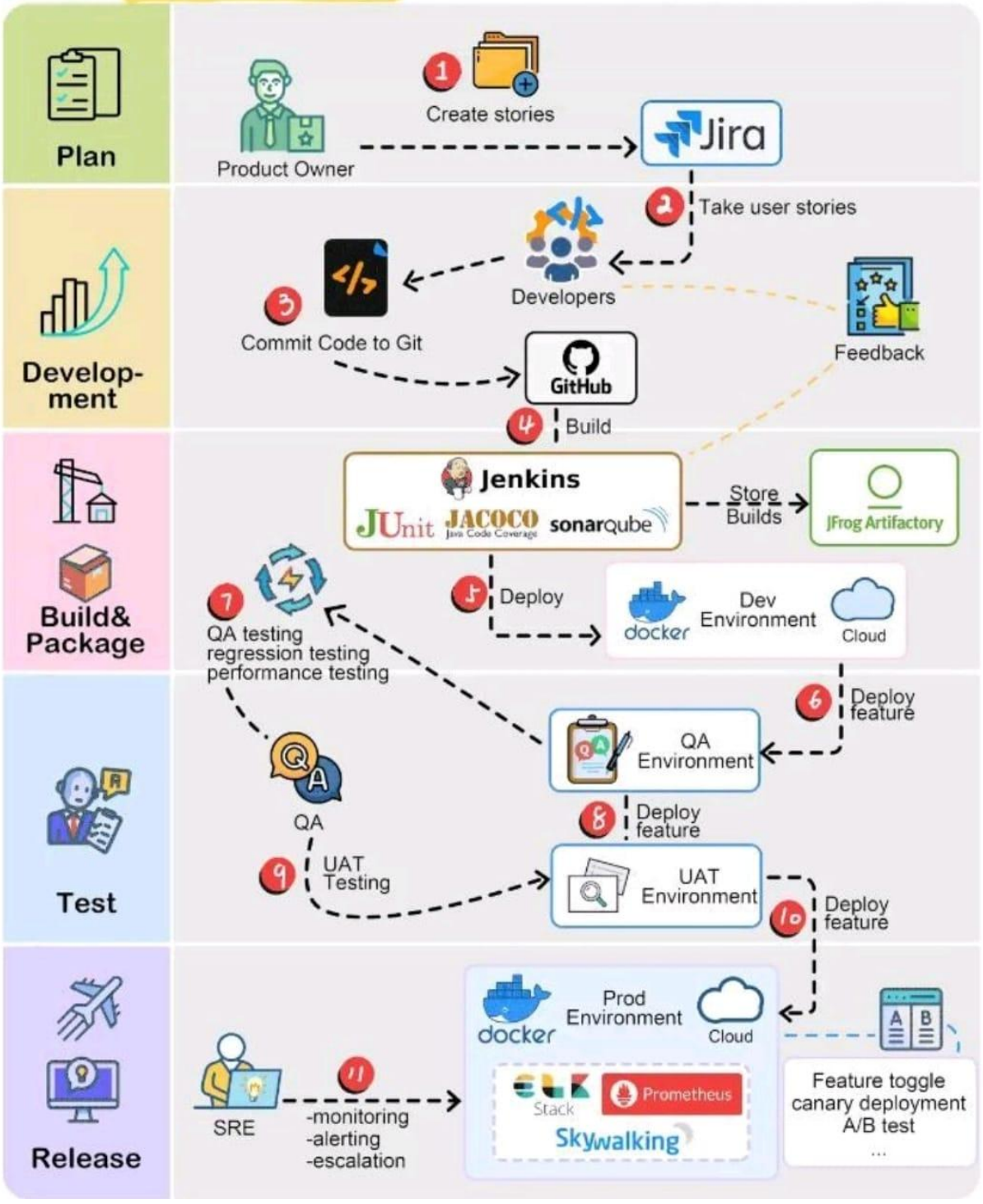


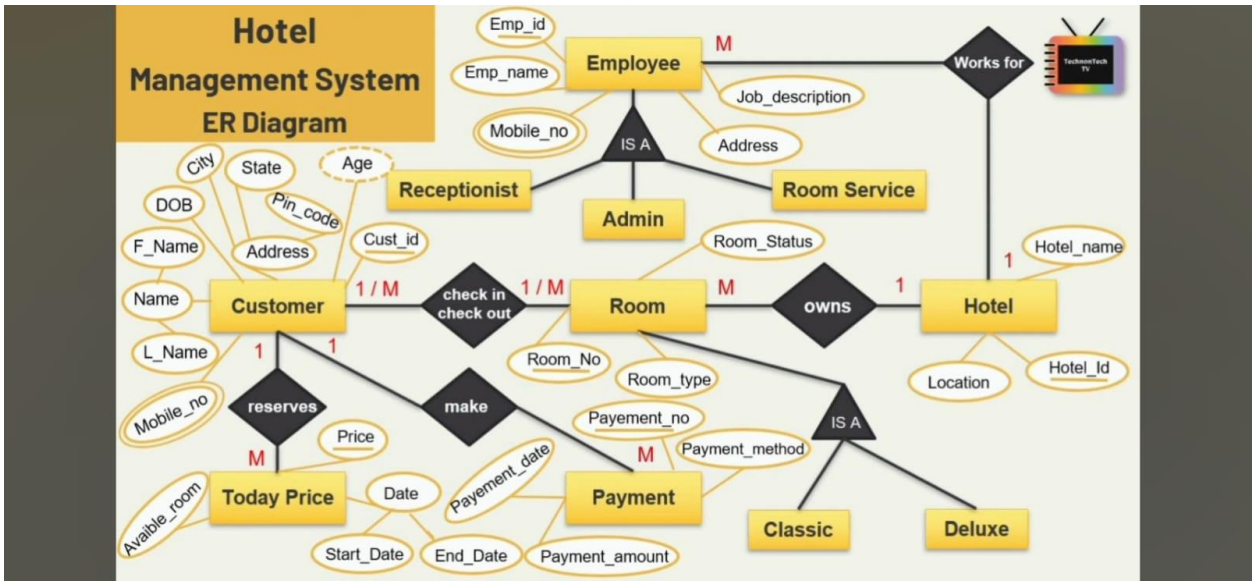

Laravel : Eloquent Model Conventions
users_data
users
App/Models/User.php
User_id
Id
Name
Age
Salary
created_at
updated_at
1
Yahoo Baba
19
4500
2024-01-01 09:00:00
2024-01-01 09:00:00
2
Salman Khan
18
5200
2024-01-01 09:00:00
2024-01-01 09:00:00
3
Anil Kapoor
20
6300
2024-01-01 09:00:00
2024-01-01 09:00:00
1. Primary key column should be named "Id"
2. Datatype Integer
3. Auto-increment
1. Necessary Columns
2. Create row date Column : created_at
3. Update row date column : updated_at
4. Date Format : timestamp
Yahoo Baba
www.yahoobaba.net

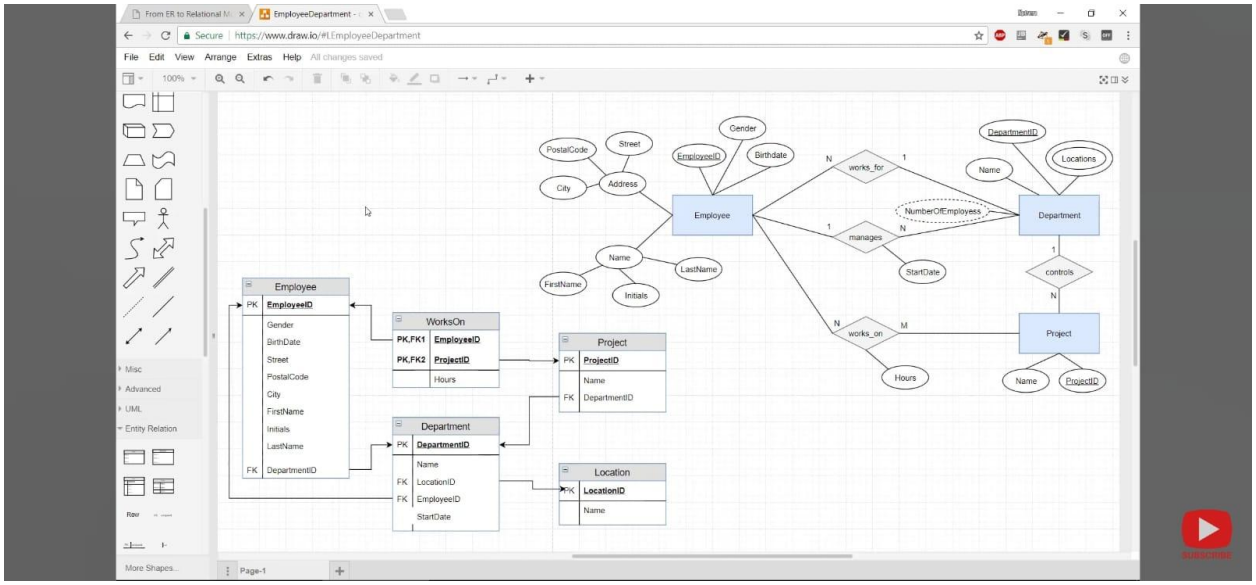

Fill in the Meta Fields
Manage Units
Create Listing Templates
Rooms Post Type
Booking Table
Booking Form
Single Page Template + Form
Date Range Filter
Orders Post Type
Connect it to the Booking Table
Add it to the Archive Page with Rooms Listing Grid

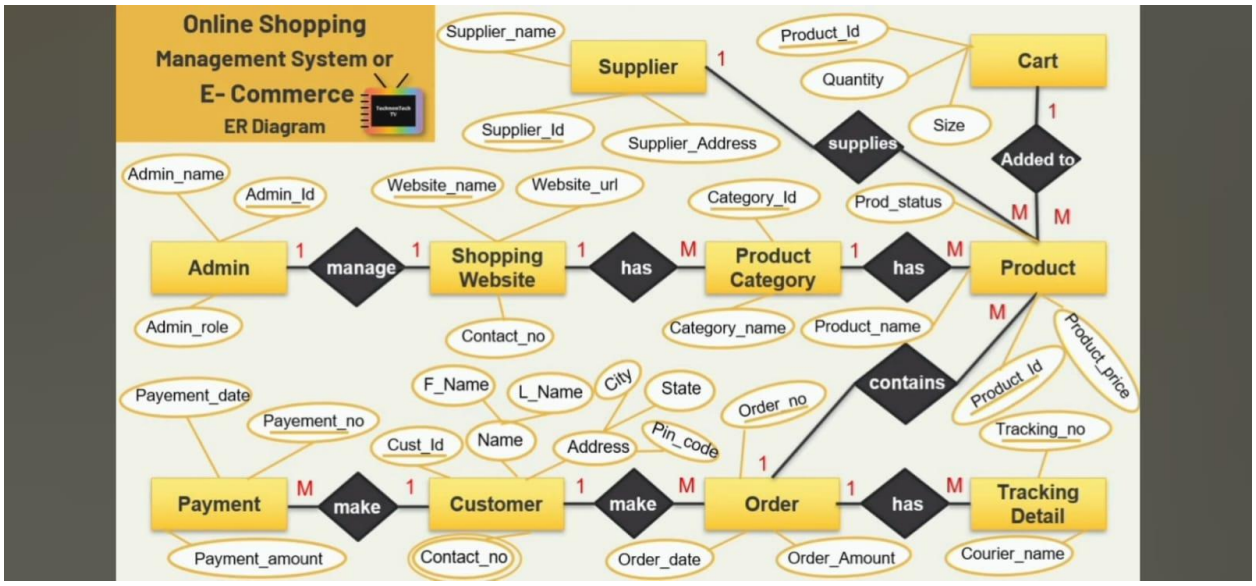
How Companies Ship Code To Production

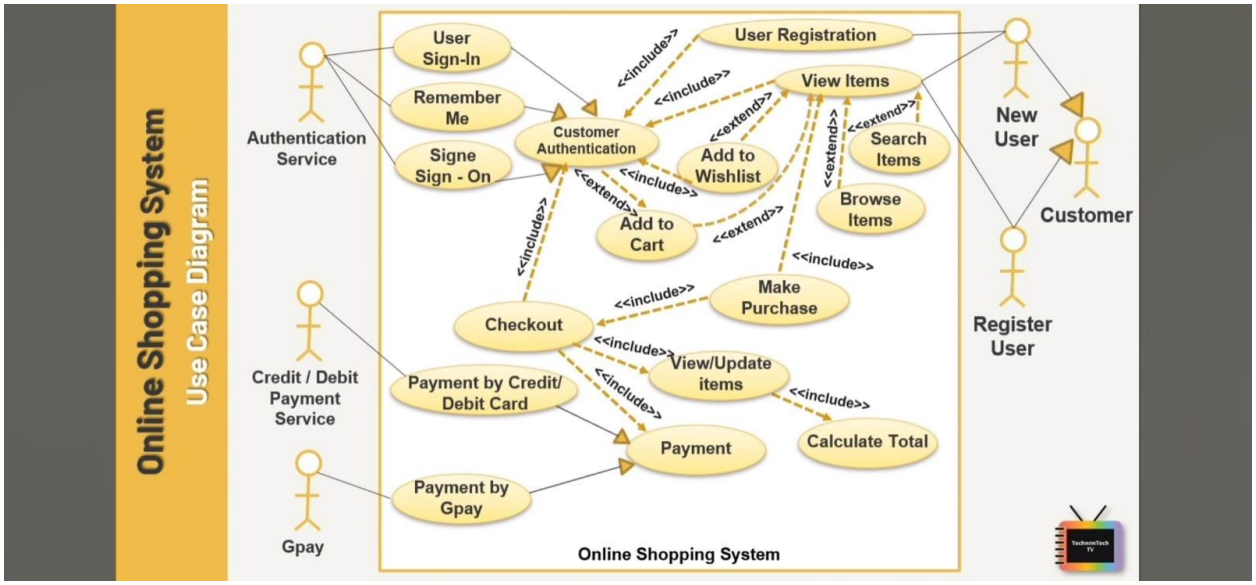
blog.bytebytego.com




Hotel
Management System
ER Diagram
Emp_id
Emp_name
Employee
M
Works for
Job_description
Mobile_no
IS A
Address
City
State
Age
Receptionist
Room Service
DOB
Pin_code
Admin
F_Name
Address
Cust_id
Room_Status
Hotel_name
Name
Customer
1 / M
check in
check out
1 / M
Room
M
owns
1
Hotel
L_Name
1
1
Room_No
Room_type
Location
Hotel_Id
Mobile_no
reserves
make
Payement_no
IS A
Price
M
M
Payment_method
Avaible_room
Today Price
Date
Payement_date
Payment
Classic
Deluxe
Start_Date
End_Date
Payment_amount


Employee
PK EmployeeID
Gender
BirthDate
Street
PostalCode
City
FirstName
Initials
LastName
FK DepartmentID
WorksOn
PK,FK1 EmployeeID
PK,FK2 ProjectID
Hours
Project
PK ProjectID
Name
FK DepartmentID
Department
PK DepartmentID
Name
FK LocationID
FK EmployeeID
StartDate
Location
PK LocationID
Name
PostalCode
Street
City
Address
Gender
EmployeeID
Birthdate
works_for
DepartmentID
Locations
Name
Department
NumberOfEmployees
manages
StartDate
controls
works_on
Hours
Project
Name
ProjectID


Online Shopping
Management System or
E- Commerce
ER Diagram
Supplier_name
Supplier
Supplier_Id
Supplier_Address
Product_Id
Quantity
Cart
Size
supplies
Added to
Admin_name
Admin_Id
Admin
manage
Website_name
Website_url
Shopping Website
has
Category_Id
Prod_status
Product Category
has
Product
Admin_role
Contact_no
Category_name
Product_name
Product_price
contains
Product_Id
Tracking_no
Payement_date
Payement_no
F_Name
L_Name
City
State
Cust_Id
Name
Address
Pin_code
Order_no
Payment
make
Customer
make
Order
has
Tracking Detail
Payment_amount
Contact_no
Order_date
Order_Amount
Courier_name


Online Shopping System
Use Case Diagram
Authentication Service
Credit / Debit Payment Service
Gpay
User Sign-In
Remember Me
Signe Sign - On
Customer Authentication
User Registration
View Items
Add to Wishlist
Search Items
Browse Items
Add to Cart
Make Purchase
Checkout
View/Update items
Payment by Credit/ Debit Card
Payment
Calculate Total
Payment by Gpay
<<include>>
<<extend>>
New User
Customer
Register User
Online Shopping System

Laraval Crud

Chatgpt Crud

```
use App\Models\YourModel;

public function store(Request $request)
{
    $validatedData = $request->validate([
        'field1' => 'required',
        'field2' => 'required',
        // Add validation rules for other fields as needed
    ]);

    YourModel::create($validatedData);

    return redirect()->route('your.route.name')->with('success',
'Record created successfully!');
}
```

```
use App\Models\YourModel;

public function index()
{
    $records = YourModel::all(); // Retrieve all records
    // You can also use where, find, or other methods to retrieve
specific records

    return view('your.view.name', compact('records'));
}
```

```
use App\Models\YourModel;

public function update(Request $request, $id)
{
    $validatedData = $request->validate([
        'field1' => 'required',
        'field2' => 'required',
    ]);
```

```
    YourModel::where('id', $id)->update($validatedData);

    return redirect()->route('your.route.name')->with('success',
'Record updated successfully!');
}
```

```
use App\Models\YourModel;

public function destroy($id)
{
    YourModel::findOrFail($id)->delete();

    return redirect()->route('your.route.name')->with('success',
'Record deleted successfully!');
}
```

```
//#####################################################
use App\Models\YourModel;
use Illuminate\Http\Request;

public function store(Request $request)
{
    $validatedData = $request->validate([
        'name' => 'required|string',
        'email' => 'required|email',
        'mobile_number' => 'required|string',
    ]);

    YourModel::create($validatedData);

    return redirect()->route('your.route.name')->with('success',
'Record created successfully!');
}
```

```
use App\Models\YourModel;
use Illuminate\Http\Request;
```

```php
public function update(Request $request, $id)
{
    $validatedData = $request->validate([
        'name' => 'required|string',
        'email' => 'required|email',
        'mobile_number' => 'required|string',
    ]);

    YourModel::where('id', $id)->update($validatedData);

    return redirect()->route('your.route.name')->with('success',
'Record updated successfully!');
}



//############################################################
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

public function store(Request $request)
{
    $validatedData = $request->validate([
        'name' => 'required|string',
        'email' => 'required|email',
        'mobile_number' => 'required|string',
    ]);

    DB::table('your_table')->insert($validatedData);

    return redirect()->route('your.route.name')->with('success',
'Record created successfully!');
}


use Illuminate\Support\Facades\DB;

public function index()
{
    $records = DB::table('your_table')->get();

    return view('your.view.name', compact('records'));
}
```

```php
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

public function update(Request $request, $id)
{
    $validatedData = $request->validate([
        'name' => 'required|string',
        'email' => 'required|email',
        'mobile_number' => 'required|string',
    ]);

    DB::table('your_table')->where('id', $id)->update($validatedData);

    return redirect()->route('your.route.name')->with('success',
'Record updated successfully!');
}



use Illuminate\Support\Facades\DB;

public function destroy($id)
{
    DB::table('your_table')->where('id', $id)->delete();

    return redirect()->route('your.route.name')->with('success',
'Record deleted successfully!');
}



//#####################################################################

<?php
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

public function store(Request $request)
{
    $validatedData = $request->validate([
        'name' => 'required|string',
        'email' => 'required|email',
        'mobile_number' => 'required|string',
        'image' => 'required|image|mimes:jpeg,png,jpg,gif|max:2048',
    ]);
```

```
    $imagePath = $request->file('image')->store('images');

    $data = array_merge($validatedData, ['image' => $imagePath]);

    DB::table('your_table')->insert($data);

    return redirect()->route('your.route.name')->with('success',
'Record created successfully!');
}


use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

public function update(Request $request, $id)
{
    $validatedData = $request->validate([
        'name' => 'required|string',
        'email' => 'required|email',
        'mobile_number' => 'required|string',
        'image' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
    ]);

    if ($request->hasFile('image')) {
        $imagePath = $request->file('image')->store('images');
        $validatedData['image'] = $imagePath;
    }

    DB::table('your_table')->where('id', $id)->update($validatedData);

    return redirect()->route('your.route.name')->with('success',
'Record updated successfully!');
}


use Illuminate\Support\Facades\DB;

public function index()
{
    $records = DB::table('your_table')->get();

    return view('your.view.name', compact('records'));
}
```

```php
use Illuminate\Support\Facades\DB;

public function destroy($id)
{
    $record = DB::table('your_table')->where('id', $id)->first();

    if ($record) {
        Storage::delete($record->image); // Delete associated image
file
        DB::table('your_table')->where('id', $id)->delete();
        return redirect()->route('your.route.name')->with('success',
'Record deleted successfully!');
    } else {
        return redirect()->route('your.route.name')->with('error',
'Record not found!');
    }
}



//#####################################################################
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\Post;
class PostController extends Controller
{
  /**
   * Display a listing of the resource.
   *
   * @return \Illuminate\Http\Response
   */
  public function index()
  {
    $posts = Post::all();
    return view('posts.index', compact('posts'));
  }
  /**
   * Store a newly created resource in storage.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return \Illuminate\Http\Response
   */
  public function store(Request $request)
  {
    $request->validate([
```

```
    'title' => 'required|max:255',
    'body' => 'required',
  ]);
  Post::create($request->all());
  return redirect()->route('posts.index')
    ->with('success', 'Post created successfully.');
}
/**
 * Update the specified resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
  $request->validate([
    'title' => 'required|max:255',
    'body' => 'required',
  ]);
  $post = Post::find($id);
  $post->update($request->all());
  return redirect()->route('posts.index')
    ->with('success', 'Post updated successfully.');
}
/**
 * Remove the specified resource from storage.
 *
 * @param  int  $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
  $post = Post::find($id);
  $post->delete();
  return redirect()->route('posts.index')
    ->with('success', 'Post deleted successfully');
}
// routes functions
/**
 * Show the form for creating a new post.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
```

```
    return view('posts.create');
  }
  /**
   * Display the specified resource.
   *
   * @param  int  $id
   * @return \Illuminate\Http\Response
   */
  public function show($id)
  {
    $post = Post::find($id);
    return view('posts.show', compact('post'));
  }
  /**
   * Show the form for editing the specified post.
   *
   * @param  int  $id
   * @return \Illuminate\Http\Response
   */
  public function edit($id)
  {
    $post = Post::find($id);
    return view('posts.edit', compact('post'));
  }
}
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\PostController;
// returns the home page with all posts
Route::get('/', PostController::class .'@index')->name('posts.index');
// returns the form for adding a post
Route::get('/posts/create', PostController::class . '@create')-
>name('posts.create');
// adds a post to the database
Route::post('/posts', PostController::class .'@store')-
>name('posts.store');
// returns a page that shows a full post
Route::get('/posts/{post}', PostController::class .'@show')-
>name('posts.show');
// returns the form for editing a post
Route::get('/posts/{post}/edit', PostController::class .'@edit')-
>name('posts.edit');
Route::put('/posts/{post}', PostController::class .'@update')-
>name('posts.update');
Route::delete('/posts/{post}', PostController::class .'@destroy')-
>name('posts.destroy');
```

API Development in Laravel

1. **Define Routes:**

   Create routes in your `routes/api.php` file to define endpoints for your API:

```php
use App\Http\Controllers\TaskController;

Route::get('/tasks', [TaskController::class, 'index']);
Route::post('/tasks', [TaskController::class, 'store']);
Route::get('/tasks/{id}', [TaskController::class, 'show']);
Route::put('/tasks/{id}', [TaskController::class, 'update']);
Route::delete('/tasks/{id}', [TaskController::class, 'destroy']);
```

1. **Create Controller:**

   Create a controller to handle these routes. Run the following command to generate a controller:

```bash
php artisan make:controller TaskController
```

   Then, implement the controller methods for your API endpoints:

```php
namespace App\Http\Controllers;

use App\Models\Task;
use Illuminate\Http\Request;

class TaskController extends Controller
{
    public function index()
    {
        return Task::all();
    }
```

```php
    public function store(Request $request)
    {
        $validatedData = $request->validate([
            'title' => 'required|string|max:255',
            'description' => 'nullable|string',
        ]);

        $task = Task::create($validatedData);

        return response()->json($task, 201);
    }

    public function show($id)
    {
        return Task::findOrFail($id);
    }
    public function update(Request $request, $id)
    {
        $task = Task::findOrFail($id);

        $validatedData = $request->validate([
            'title' => 'required|string|max:255',
            'description' => 'nullable|string',
        ]);

        $task->update($validatedData);

        return response()->json($task, 200);
    }

    public function destroy($id)
    {
        $task = Task::findOrFail($id);
        $task->delete();

        return response()->json(null, 204);
    }
}
```

1. **Define Model**:

   Create a model for your `tasks` table:

```bash
php artisan make:model Task -m
```

   This will generate a `Task` model and a migration file. Define the schema for your `tasks` table in the migration file and then run the migration to create the table:

```bash
php artisan migrate
```

   Ensure your `Task` model has the correct fillable properties:

Ensure your `Task` model has the correct fillable properties:

```php
namespace App\Models;

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

class Task extends Model
{
    use HasFactory;

    protected $fillable = ['title', 'description'];
}
```

Step 1 : Install Laravel

```
composer create-project --prefer-dist laravel/laravel myFstLaravel
```

Step 2: Update Database Configuration

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=LaravelDemo
DB_USERNAME=root
DB_PASSWORD=
```

Set the above value in .env file. make sure you mysql is runnig and LaravelDemo db exists.

Step 3: Create Table

```
php artisan make:migration –help
```

Syntax

```
php artisan make:migration <fileName> --create=<tableName>
```

Example

```
php artisan make:migration user_data --create=user_data
```

or

```
php artisan make:migration --path=/database/migrations/user_data
```

user_data --create=user_data migration file created on user_data folder.

put the table field in user_data migration file

path = database/migration/

```
$table->increments('id');
$table->string('name');
$table->string('email');
$table->timestamps();
```

save the file and execute one more command to migrate table to db

```
php artisan migrate
```

or

```
php artisan migrate --path=/database/migrations/user_data/
```

Step 4: Create Resource Route

Create Resource Route or create Route.

Actually create Resource route give us shortcut to create route

example

Route::resource('/userData','UserDataController');

we can see routes in terminal.

```
php artisan route : list
```

Step 5: Create Controller and Model

```
php artisan help make:controller
```

It'll show the list of argument.

To create controller and model just execute the following command.

```
php artisan make:controller UserDataController -r -m UserData
```

terminal will ask : A App\UserData model does not exist. Do you want to generate it? (yes/no) [yes]: type yes and press enter.

Step 6: Create Blade Files

UserDataController.php

```php
<?php

namespace App\Http\Controllers;

use App\UserData;
use Illuminate\Http\Request;

class UserDataController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
```

```
    public function index()
    {
        $userData = UserData::latest()->paginate(5);

        return view('userData.index',compact('userData'))
            ->with('i', (request()->input('page', 1) - 1) * 5);
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        return view('userData.create');
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required',
            'email' => 'required',
        ]);

        UserData::create($request->all());

        return redirect()->route('userData.index')
                        ->with('success','User created
successfully.');
    }

    /**
     * Display the specified resource.
     *
     * @param  \App\UserData  $userData
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        $userData = UserData::find($id);
```

```
        return view('userData.show',compact('userData'));
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  \App\UserData  $userData
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        $userData = UserData::find($id);
        return view('userData.edit',compact('userData','id'));
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \App\UserData  $userData
     * @return \Illuminate\Http\Response
     */
    public function update($id)
    {
        $userData = UserData::find($id);
        $userData->name = request('name');
        $userData->email = request('email');
        $userData->save();
                $request->validate([
                'name' => 'required',
                'email' => 'required',
         ]);
        $userData->update($request->all());

        return redirect()->route('userData.index')
                        ->with('success','User updated
successfully');
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  \App\UserData  $userData
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
```

```
        UserData::find($id)->delete();

        return redirect()->route('userData.index')
                         ->with('success','User deleted
successfully');
    }
}
```

UserData.php (modal)

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class UserData extends Model
{
    protected $table = 'user_data';
    protected $fillable = [
        'name', 'email'
    ];
}
```

layout.blade.php

```
<!DOCTYPE html>
<html lang="{{ app()->getLocale() }}">
<head>
    <title>LaravelDemo</title>
    <!-- Fonts -->
    <link
href="https://fonts.googleapis.com/css?family=Raleway:100,600"
rel="stylesheet" type="text/css">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootst
rap.min.css">

    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.m
in.js"></script>
```

```
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstra
p.min.js"></script>
    <!-- Style -->

</head>
<body>

<div class="container">
    @yield('content')
</div>

</body>
</html>
```

index.blade.php

```
@extends('userData.layout')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>Laravel Demo</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-success" href="{{
route('userData.create') }}"> Create New User</a>
            </div>
        </div>
    </div>

    @if ($message = Session::get('success'))
        <div class="alert alert-success">
            <p>{{ $message }}</p>
        </div>
    @endif

    <table class="table table-bordered">
        <tr>
            <th>No</th>
            <th>Name</th>
            <th>email</th>
            <th width="280px">Action</th>
        </tr>
        @foreach ($userData as $uData)
```

```
        <tr>
            <td>{{ ++$i }}</td>
            <td>{{ $uData->name }}</td>
            <td>{{ $uData->email }}</td>
            <td>
                <form action="{{
route('userData.destroy',$uData->id) }}" method="POST">

                    <a class="btn btn-info" href="{{
route('userData.show',$uData->id) }}">Show</a>



                    <a class="btn btn-primary" href="{{
route('userData.edit',$uData->id) }}">Edit</a>
                    <!-- SUPPORT ABOVE VERSION 5.5 -->
                    {{-- @csrf
                    @method('DELETE') --}}

                    {{ csrf_field() }}
                    {{ method_field('DELETE') }}


                    <button type="submit" class="btn btn-
danger">Delete</button>
                </form>
            </td>
        </tr>
        @endforeach
    </table>

    {!! $userData->links() !!}

@endsection
```

create.blade.php

```
@extends('userData.layout')

@section('content')
<div class="row">
    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2>Add New User</h2>
        </div>
        <div class="pull-right">
```

```
            <a class="btn btn-primary" href="{{
route('userData.index') }}"> Back</a>
        </div>
    </div>
</div>

@if ($errors->any())
    <div class="alert alert-danger">
        <strong>Whoops!</strong> There were some problems with
your input.<br><br>
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif

<form action="{{ route('userData.store') }}" method="POST">
    {{ csrf_field() }}

     <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Name:</strong>
                <input type="text" name="name" class="form-
control" placeholder="Name">
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Email Id:</strong>
                <input type="text" class="form-control"
name="email" placeholder="Email Id">
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12 text-center">
                <button type="submit" class="btn btn-
primary">Submit</button>
        </div>
    </div>

</form>
@endsection
```

edit.blade.php

```
@extends('userData.layout')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>Edit User</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-primary" href="{{
route('userData.index') }}"> Back</a>
            </div>
        </div>
    </div>

    @if ($errors->any())
        <div class="alert alert-danger">
            <strong>Whoops!</strong> There were some problems
with your input.<br><br>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif

    <form action="{{ route('userData.update',$userData->id) }}"
method="POST">
        {{ csrf_field() }}
        {{ method_field('PATCH') }}

         <div class="row">
            <div class="col-xs-12 col-sm-12 col-md-12">
                <div class="form-group">
                    <strong>Name:</strong>
                    <input type="text" name="name" value="{{
$userData->name }}" class="form-control" placeholder="Name">
                </div>
            </div>
            <div class="col-xs-12 col-sm-12 col-md-12">
                <div class="form-group">
                    <strong>Email ID:</strong>
                    <input type="text" class="form-control"
name="email" value ="{{ $userData->email }}" placeholder="Email
ID">
```

```
                </div>
            </div>
            <div class="col-xs-12 col-sm-12 col-md-12 text-
center">
              <button type="submit" class="btn btn-
primary">Submit</button>
            </div>
        </div>

    </form>
@endsection
```

show.blade.php

```
@extends('userData.layout')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2> Show User</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-primary" href="{{
route('userData.index') }}"> Back</a>
            </div>
        </div>
    </div>

    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Name:</strong>
                {{ $userData->name }}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Details:</strong>
                {{ $userData->email }}
            </div>
        </div>
    </div>
@endsection
```

How to insert data in database

```
CREATE TABLE student_details
(
id int NOT NULL AUTO_INCREMENT,
first_name varchar(50),
last_name varchar(50),
city_name varchar(50),
email varchar(50),
PRIMARY KEY (id)
);
```

Create 3 files for insert data in Laravel :

- **StudInsertController.php (app/Http/Controllers/StudInsertController.php)**
- **stud_create.php (resources/views/stud_create.php)**
- **web.php (routes/web.php)**

stud_create.blade.php(View File)

```
<!DOCTYPE html>
<html>
<head>
<title>Student Management | Add</title>
</head>
<body>
@if (session('status'))
<div class="alert alert-success" role="alert">
      <button type="button" class="close" data-
dismiss="alert">×</button>
      {{ session('status') }}
</div>
@elseif(session('failed'))
<div class="alert alert-danger" role="alert">
      <button type="button" class="close" data-
dismiss="alert">×</button>
      {{ session('failed') }}
</div>
@endif
<form action = "/create" method = "post">
```

```
        <input type = "hidden" name = "_token" value = "<?php echo
csrf_token(); ?>">
        <table>
        <tr>
        <td>First Name</td>
        <td><input type='text' name='first_name' /></td>
        <tr>
        <td>Last Name</td>
        <td><input type="text" name='last_name'/></td>
        </tr>
        <tr>
        <td>City Name</td>
        <td>
        <select name="city_name">
        <option value="bbsr">Bhubaneswar</option>
        <option value="cuttack">Cuttack</option>
        </select></td>
        </tr>
        <tr>
        <td>Email</td>
        <td><input type="text" name='email'/></td>
        </tr>

        <tr>
        <td colspan = '2'>
        <input type = 'submit' value = "Add student"/>
        </td>
        </tr>
        </table>
</form>
</body>
</html>
```

StudInsertController.php(Controller File)

```
<?php
namespace App\Http\Controllers;
use App\StudInsert;
use Illuminate\Http\Request;
use \Illuminate\Http\Response;
use Illuminate\Support\Facades\Validator;
use Illuminate\Validation\Rule;

class StudInsertController extends Controller
{
```

```
    public function insert(){
        $urlData = getURLList();
        return view('stud_create');
    }
    public function create(Request $request){
        $rules = [
                    'first_name' =>
'required|string|min:3|max:255',
                    'city_name' =>
'required|string|min:3|max:255',
                    'email' => 'required|string|email|max:255'
            ];
            $validator = Validator::make($request-
>all(),$rules);
            if ($validator->fails()) {
                  return redirect('insert')
                  ->withInput()
                  ->withErrors($validator);
            }
            else{
           $data = $request->input();
                  try{
                        $student = new StudInsert;
               $student->first_name = $data['first_name'];
               $student->last_name = $data['last_name'];
                        $student->city_name =
$data['city_name'];
                        $student->email = $data['email'];
                        $student->save();
                        return redirect('insert')-
>with('status',"Insert successfully");
                  }
                  catch(Exception $e){
                        return redirect('insert')-
>with('failed',"operation failed");
                  }
            }
    }
}
```

StudInsert.php(Model File)

```php
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class StudInsert extends Model
{
	protected $table = 'student_details';
	public $timestamps = true;
	/**
	 * The attributes that are mass assignable.
	 *
	 * @var array
	 */
	protected $fillable = [
		'first_name', 'last_name','city_name', 'email',
	];
}
```

web.php

```php
<?php
//insert data
Route::get('insert','StudInsertController@insertform');
Route::post('create','StudInsertController@insert');
```

Retrieve data from database

Create 3 files for retrieve data in Laravel :

- **StudViewController.php (app/Http/Controllers/StudViewController.php)**
- **stud_view.blade.php (resources/views/stud_view.blade.php)**
- **web.php (routes/web.php)**

stud_view.blade.php

```
<!DOCTPE html>
<html>
<head>
<title>View Student Records</title>
</head>
<body>
<table border = "1">
<tr>
<td>Id</td>
<td>First Name</td>
<td>Last Name</td>
<td>City Name</td>
<td>Email</td>
</tr>
@foreach ($users as $user)
<tr>
<td>{{ $user->id }}</td>
<td>{{ $user->first_name }}</td>
<td>{{ $user->last_name }}</td>
<td>{{ $user->city_name }}</td>
<td>{{ $user->email }}</td>
</tr>
@endforeach
</table>
</body>
</html>
```

StudViewController.php

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use DB;
use App\Http\Requests;
use App\Http\Controllers\Controller;

class StudViewController extends Controller {

public function index(){
$users = DB::select('select * from student_details');
return view('stud_view',['users'=>$users]);
}
}
```

web.php

```
Route::get('view-records','StudViewController@index');
```

Update data from database

Create 4 file for update data

- **StudUpdateController.php (app/Http/Controllers/StudUpdateController.php)**
- **stud_edit_view.blade.php (resources/views/stud_edit_view.blade.php)**
- **stud_update.php (resources/views/stud_update.php)**
- **web.php (routes/web.php)**

stud_edit_view.blade.php

```
<!Doctype html>
<html>
<head>
<title>View Student Records</title>
</head>
<body>
<table border = "1">
<tr>
<td>ID</td>
<td>First Name</td>
<td>Lastst Name</td>
<td>City Name</td>
<td>Email</td>
<td>Edit</td>
</tr>
@foreach ($users as $user)
<tr>
<td>{{ $user->id }}</td>
<td>{{ $user->first_name }}</td>
<td>{{ $user->last_name }}</td>
<td>{{ $user->city_name }}</td>
<td>{{ $user->email }}</td>
<td><a href = 'edit/{{ $user->id }}'>Edit</a></td>
</tr>
```

```
@endforeach
</table>
</body>
</html>
```

stud_update.php

```
<!DOCTYPE html>
<html>
<head>
<title>Student Management | Edit</title>
</head>
<body>
<form action = "/edit/<?php echo $users[0]->id; ?>" method = "post">
<input type = "hidden" name = "_token" value =
"<?php echo csrf_token(); ?>">
<table>
<tr>
<td>First Name</td>
<td>
<input type = 'text' name = 'first_name'
value = '<?php echo$users[0]->first_name; ?>'/> </td>
</tr>
<tr>
<td>Last Name</td>
<td>
<input type = 'text' name = 'last_name'
value = '<?php echo$users[0]->last_name; ?>'/>
</td>
</tr>
<tr>
<td>City Name</td>
<td>
<input type = 'text' name = 'city_name'
value = '<?php echo$users[0]->city_name; ?>'/>
</td>
</tr>
<tr>
<td>Email</td>
<td>
<input type = 'text' name = 'email'
```

```
value = '<?php echo$users[0]->email; ?>'/>
</td>
</tr>
<tr>
<td colspan = '2'>
<input type = 'submit' value = "Update student" />
</td>
</tr>
</table>
</form>
</body>
</html>
```

StudUpdateController.php

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use DB;
use App\Http\Requests;
use App\Http\Controllers\Controller;
class StudUpdateController extends Controller {
public function index(){
$users = DB::select('select * from student');
return view('stud_edit_view',['users'=>$users]);
}
public function show($id) {
$users = DB::select('select * from student where id = ?',[$id]);
return view('stud_update',['users'=>$users]);
}
public function edit(Request $request,$id) {
$first_name = $request->input('first_name');
$last_name = $request->input('last_name');
$city_name = $request->input('city_name');
$email = $request->input('email');
```

```
DB::update('update student set first_name =
?,last_name=?,city_name=?,email=? where id =
?',[$first_name,$last_name,$city_name,$email,$id]);
echo "Record updated successfully.
";
echo 'Click Here to go back.';
}
}


Route::get('edit-records','StudUpdateController@index');
Route::get('edit/{id}','StudUpdateController@show');
Route::post('edit/{id}','StudUpdateController@edit');
```

How to delete data from database

Create 3 file for delete data

- **StudDeleteController.php (app/Http/Controllers/StudDeleteController.php)**
- **stud_delete_view.blade.php (resources/views/stud_delete_view.blade.php)**
- **web.php (routes/web.php)**

stud_delete_view.blade.php

```
<!DOCTPE html>
<html>
<head>
<title>View Student Records</title>
</head>
<body>
<table border = "1">
<tr>
<td>ID</td>
<td>First Name</td>
<td>Last Name</td>
<td>City Name</td>
<td>Email</td>
```

```
<td>Edit</td>
</tr>
@foreach ($users as $user)
<tr>
<td>{{ $user->id }}</td>
<td>{{ $user->first_name }}</td>
<td>{{ $user->last_name }}</td>
<td>{{ $user->city_name }}</td>
<td>{{ $user->email }}</td>
<td><a href = 'delete/{{ $user->id }}'>Delete</a></td>
</tr>
@endforeach
</table>
</body>
</html>
```

StudDeleteController.php

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use DB;
use App\Http\Requests;
use App\Http\Controllers\Controller;
class StudDeleteController extends Controller {
public function index(){
$users = DB::select('select * from student_details');
return view('stud_delete_view',['users'=>$users]);
}
public function destroy($id) {
DB::delete('delete from student_details where id = ?',[$id]);
echo "Record deleted successfully.
";
echo 'Click Here to go back.';
}
}
```

```
Route::get('delete-records','StudDeleteController@index');
Route::get('delete/{id}','StudDeleteController@destroy');
```

What is Composer in PHP?

If you've been coding in PHP for some time, you'll be aware of how **PHP libraries can help save work** and make code **reusable**. In the past, it was **harder to add libraries to PHP**.

That's where a dependency manager like **Composer comes in**. In fact, before Composer, there was a popular tool called **PEAR which was used to manage PHP extensions and libraries. But it had its own limitations**, which Composer was created to address.

Composer is a **dependency manager for PHP.**

You can simply run **composer update** to get the **latest compatible packages**.

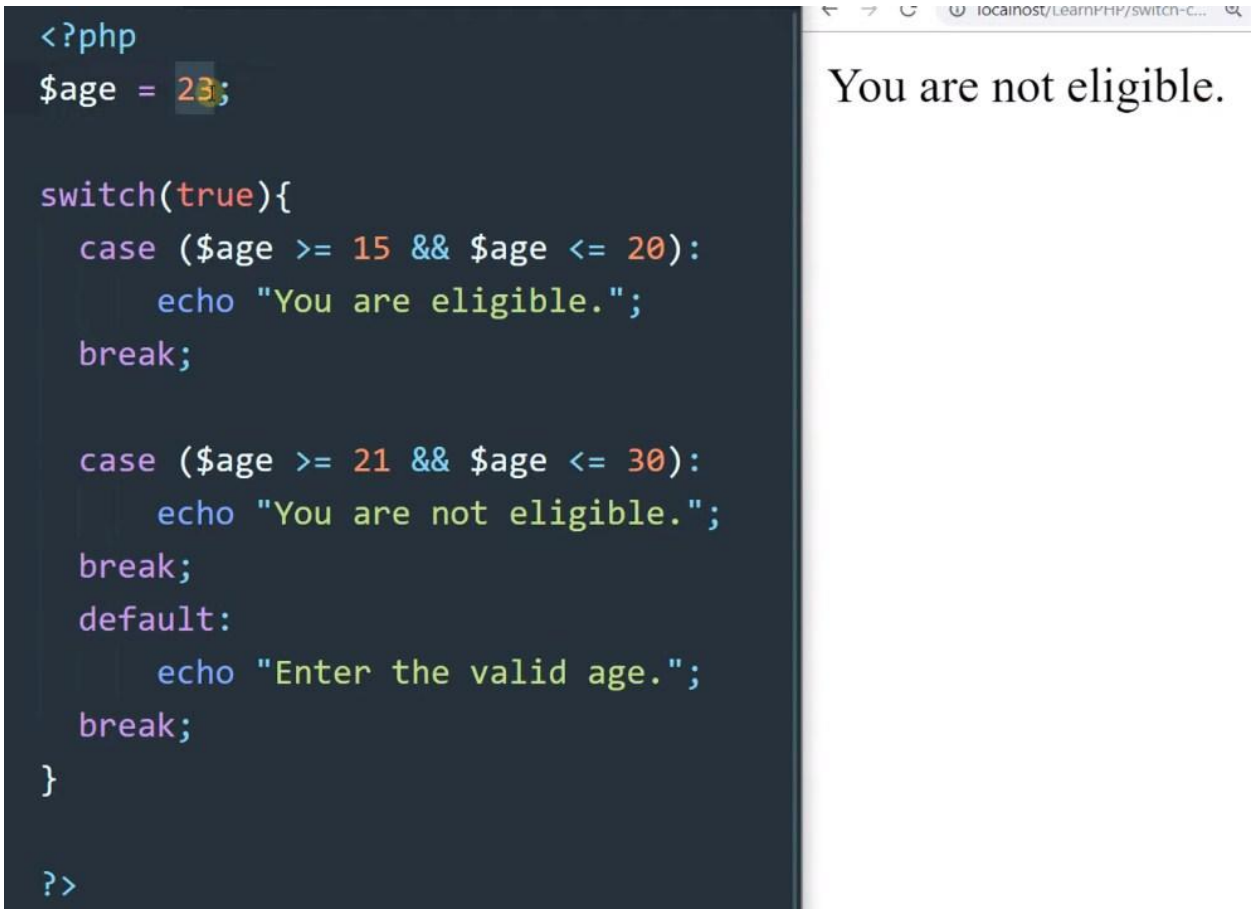
Ex

❖ To ensure that the composer is successfully installed:

$ composer

❖ To install the package:

$ composer require phpmailer


<?php
$age = 23;
switch(true){
case ($age >= 15 && $age <= 20):
echo "You are eligible.";
break;
case ($age >= 21 && $age <= 30):
echo "You are not eligible.";
break;
default:
echo "Enter the valid age.";
break;
}
?>
localhost/LearnPHP/switch-c...
You are not eligible.

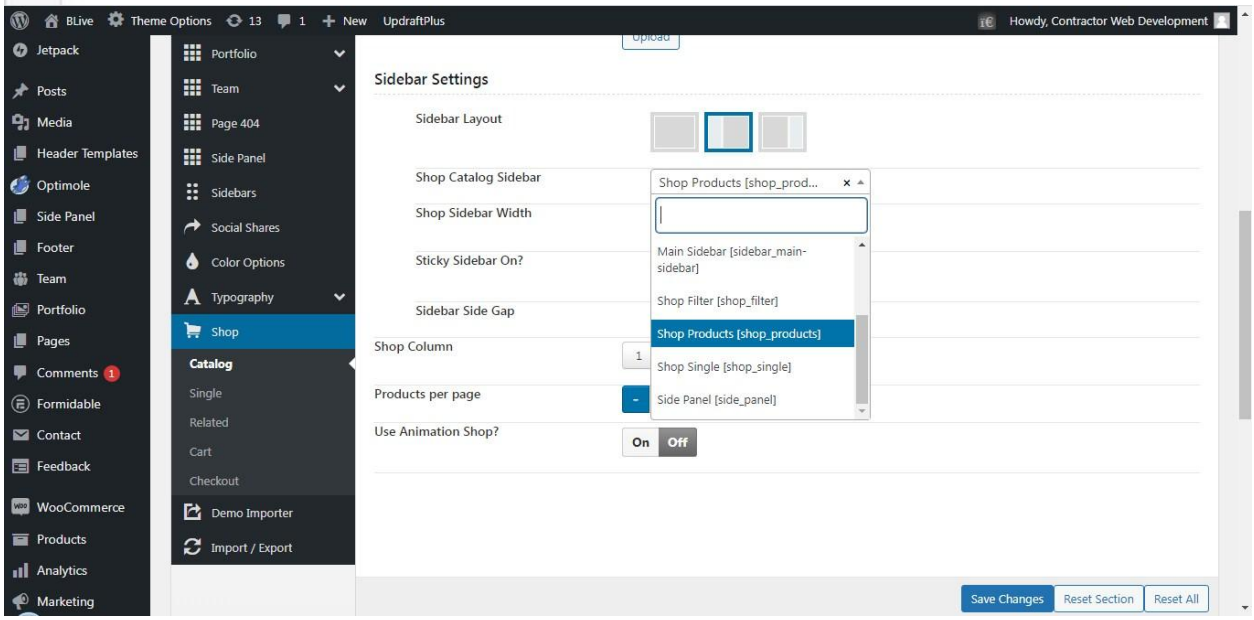
PHP Switch Example-

```php
<?php
$num=20;
switch($num){
case 10:
echo("number is equals to 10");
break;
case 20:
echo("number is equal to 20");
break;
case 30:
echo("number is equal to 30");
break;
default:
echo("number is not equal to 10, 20 or 30");
}
?>
```

```php
<?php
$day = "Monday";

switch ($day) {
    case "Monday":
        echo "Today is Monday";
        break;
    case "Tuesday":
        echo "Today is Tuesday";
        break;
    case "Wednesday":
        echo "Today is Wednesday";
        break;
    case "Thursday":
        echo "Today is Thursday";
        break;
    case "Friday":
        echo "Today is Friday";
        break;
    case "Saturday":
        echo "Today is Saturday";
        break;
    case "Sunday":
        echo "Today is Sunday";
        break;
    default:
        echo "Not a valid day";
}
?>
```

```php
<?php
$score = 85;

switch (true) {
    case ($score >= 90):
        echo "You got an A";
        break;
    case ($score >= 80 && $score < 90):
        echo "You got a B";
        break;
    case ($score >= 70 && $score < 80):
        echo "You got a C";
        break;
    case ($score >= 60 && $score < 70):
        echo "You got a D";
        break;
    default:
        echo "You got an F";
}
?>
```

```php
add_action('woocommerce_single_product_summary', 'fn_custom_block', 100);
function fn_custom_block() {
    echo '<div class="custom-block">';
    echo do_shortcode('[contact-form-7 id="4772" title="Inquiry Form"]');
    echo '</div>';
}
```



How to code in MVC Framework ?

Controller (controllers/my_controller.php)

```
function total_frogs(){
  $this->load->model("frogs");
  $number_of_frogs= $this->frogs->count_frogs();
  $data['froggies'] = $number_of_frogs;
  $this->load->view("frog_view",$data);
}
```

Model (models/frogs.php)

```
function count_frogs(){
  $this->db->where("type","frog");
  $this->db->from("animals");
  $query = $this->db->get();
  return $query->numb_rows();
}
```

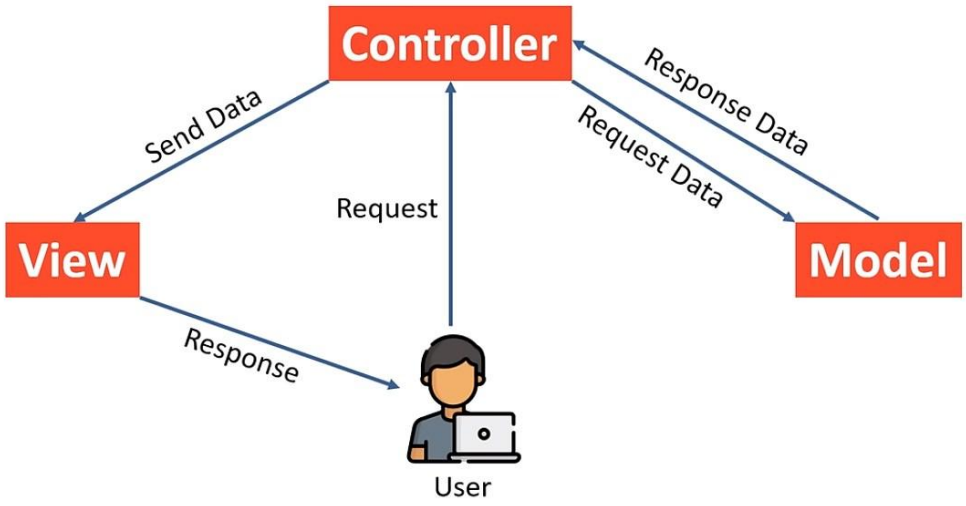
View (views/frog_view.php)

```
<html>
  <body>
    <h1>You've <?=$name;?> frogs in list</h1>
  </body>
</html>
```


Benefits of MVC Framework

- Organized Code
- Independent Block
- Reduces the complexity of Web Applications
- Easy to maintain
- Easy to modify
- Code reusability
- Improved collaboration
- Platform independence

MVC Pattern Work Flow



Popular MVC Frameworks

PHP MVC Framework

- Laravel
- Symfony
- CodeIgniter
- Yii
- CakePHP
- Zend Framework

Other Programming MVC Framework

- Django & Flask (Python)
- Ruby on Rails (Ruby)
- Express.js (JavaScript/Node.js)
- ASP.NET MVC (ASP.net Core)

What is Framework ?

Programming frameworks are sets of pre-written code and libraries that provide a foundation for developing software applications.

Pre-written Code & Library

- Tools
- Components
- Modules

Examples

- Database Component
- Caching
- Pagination
- Session management
- Form Handling
- Security mechanisms
- User authentication
- APIs
- Payment Gateways

Benefits

- Code organization
- Reusability
- Standardization
- Testing & debugging support
- Community and support

Benefits of Laravel Framework

- Open source
- Elegant syntax
- MVC architecture
- Database migration and ORM
- Robust routing system
- Command-line Interface (Composer)
- Powerful template engine (Blade Template)
- Authentication and authorization
- Testing and debugging
- Security (XSS, CSRF, SQL injection)
- Scalability and performance(Redis and Memcached)
- Robust ecosystem and community

What we will learn in this Laravel Series

- Artisan CLI
- Routing
- Views
- Blade Template
- Controllers
- Model
- Database
- Eloquent ORM
- Migration
- Middleware
- Form Validation
- Authentication
- Handling File Upload
- APIs Validation
- CRUD Project
- News Blog Project

Laravel Route

http://localhost/about

↓

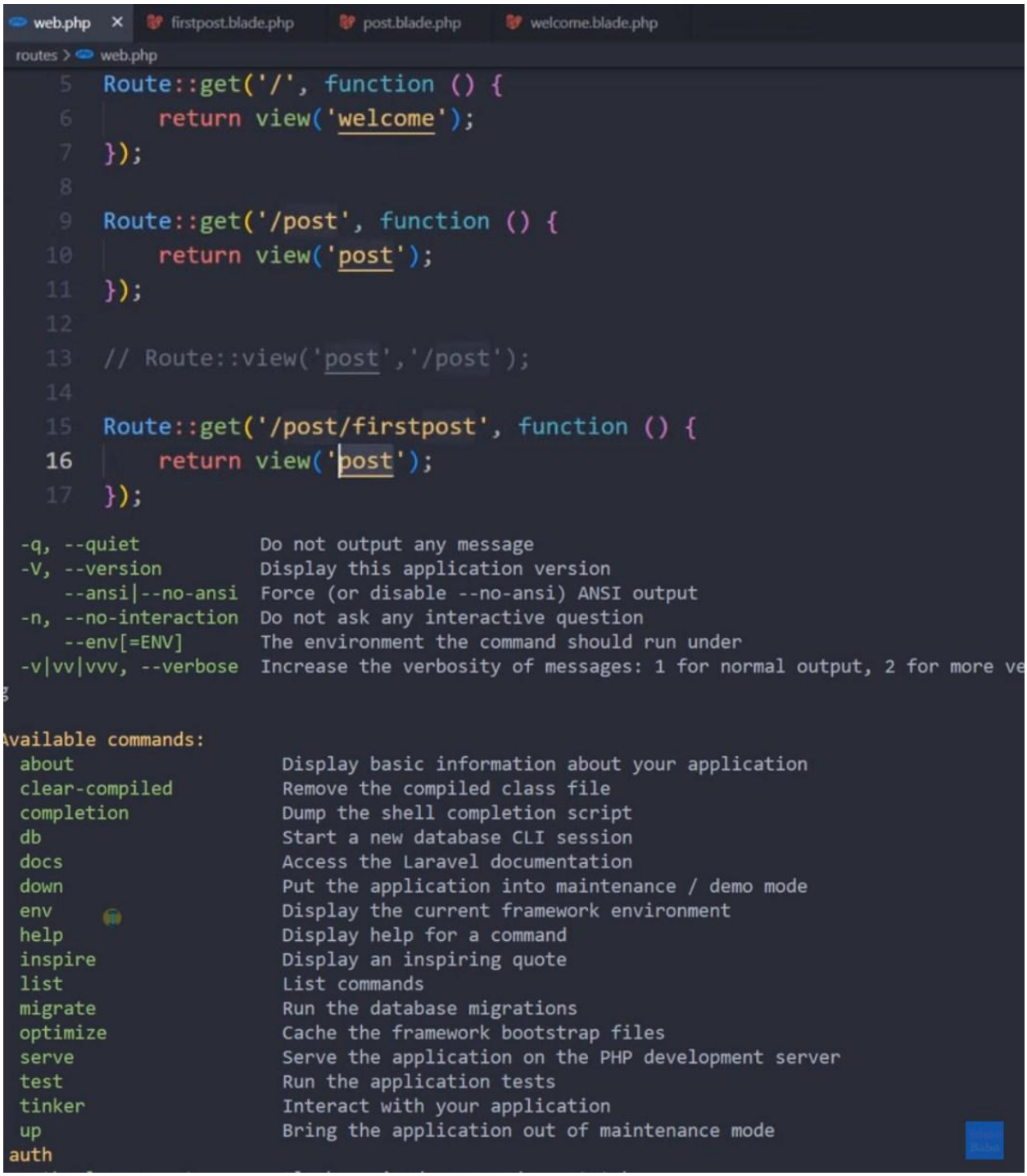
Route Name

↓

hello.php (View File / Controller File)

routes/web.php

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


web.php
firstpost.blade.php
post.blade.php
welcome.blade.php
routes > web.php
Route::get('/', function () {
return view('welcome');
});
Route::get('/post', function () {
return view('post');
});
// Route::view('post','/post');
Route::get('/post/firstpost', function () {
return view('post');
});
-q, --quiet Do not output any message
-V, --version Display this application version
--ansi|--no-ansi Force (or disable --no-ansi) ANSI output
-n, --no-interaction Do not ask any interactive question
--env[=ENV] The environment the command should run under
-v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more ve
Available commands:
about Display basic information about your application
clear-compiled Remove the compiled class file
completion Dump the shell completion script
db Start a new database CLI session
docs Access the Laravel documentation
down Put the application into maintenance / demo mode
env Display the current framework environment
help Display help for a command
inspire Display an inspiring quote
list List commands
migrate Run the database migrations
optimize Cache the framework bootstrap files
serve Serve the application on the PHP development server
test Run the application tests
tinker Interact with your application
up Bring the application out of maintenance mode
auth

```
 queue:retry-batch       Retry the failed jobs for a batch
 queue:table             Create a migration for the queue jobs database table
 queue:work              Start processing jobs on the queue as a daemon
route
 route:cache             Create a route cache file for faster route registration
 route:clear             Remove the route cache file
 route:list              List all registered routes
sail
 sail:add                Add a service to an existing Sail installation
 sail:install            Install Laravel Sail's default Docker Compose file
 sail:publish            Publish the Laravel Sail Docker files
sanctum
 sanctum:prune-expired   Prune tokens expired for more than specified number of hours
schedule
 schedule:clear-cache    Delete the cached mutex files created by scheduler
 schedule:list           List all scheduled tasks
 schedule:run            Run the scheduled commands
 schedule:test           Run a scheduled command
 schedule:work           Start the schedule worker
schema
 schema:dump             Dump the given database schema
session
 session:table           Create a migration for the session database table
storage
 storage:link            Create the symbolic links configured for the application
stub
 stub:publish            Publish all stubs that are available for customization
vendor
 vendor:publish          Publish any publishable assets from vendor packages
view
 view:cache              Compile all of the application's Blade templates
```

Laravel Route Parameters

http://localhost/post/10

http://localhost/post/yahoobaba

http://localhost/post/news10

http://localhost/post/@news10

```
Route::get('/post/{id}', function (string $id) {
   return 'User ' . $id;
});
```

web.php

```php
<?php

use Illuminate\Support\Facades\Route;

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

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

Route::view('/post','post');

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

web.php

```php
<?php

use Illuminate\Support\Facades\Route;

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

Route::get('/post/{id?}', function (string $id = null) {
    if($id){
        return "<h1>Post ID : ". $id ."</h1>";
    }else{
        return "<h1>No ID Found</h1>";
    }


});
```

Laravel Route Constraints

http://localhost/post/10 → whereNumber('id')

http://localhost/post/yahoobaba → whereAlpha('name')

http://localhost/post/news10 → whereAlphaNumeric('name')

http://localhost/post/song → whereIn('category', ['movie', 'song'])

http://localhost/post/@10 → where('id', '[@0-9]+')

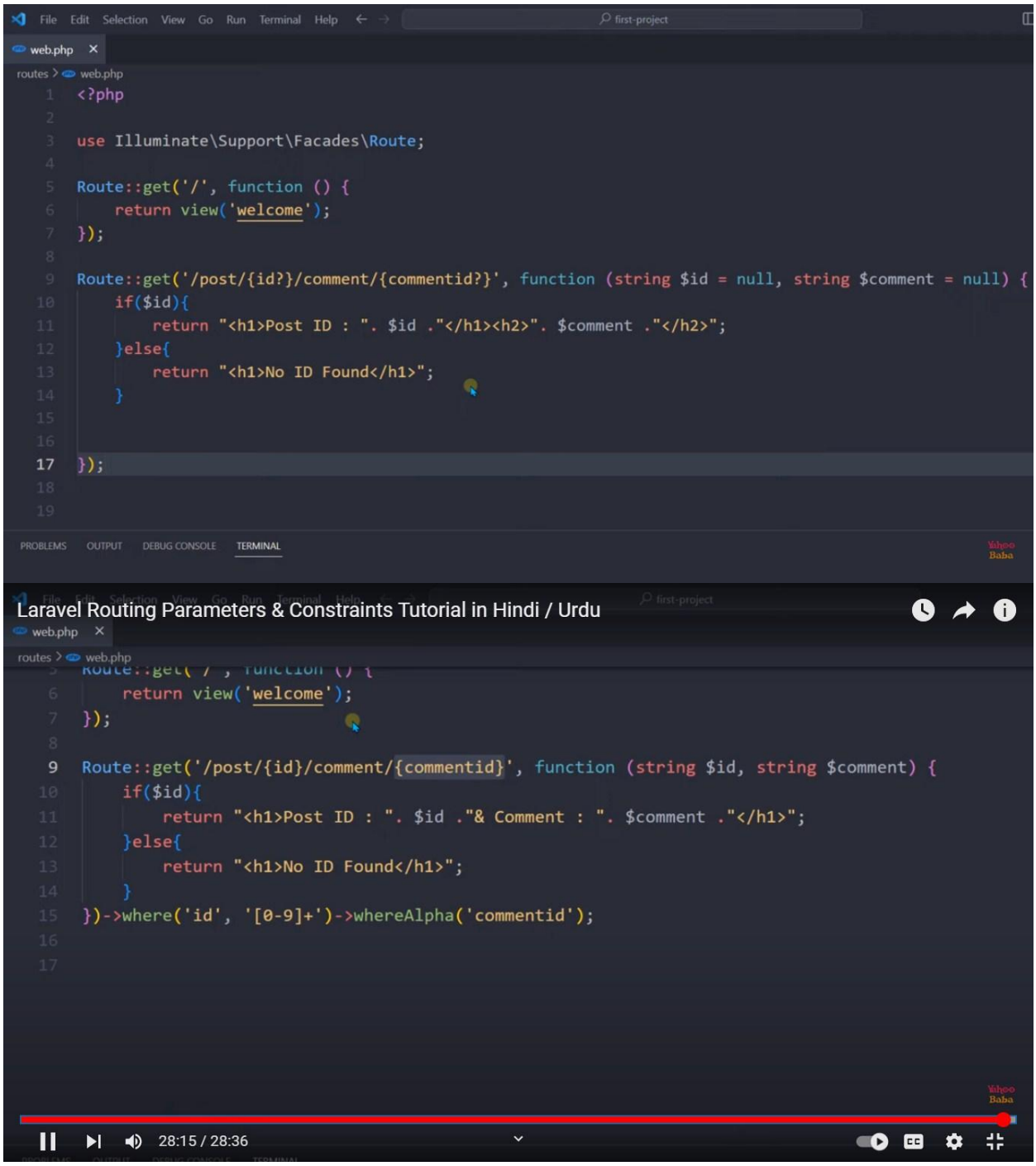
↓

Regular Expressions

Laravel Route Constraints

```
Route::get('/post/{id}', function (string $id) {
    return 'User ' . $id;
})->whereNumber('id');
```

Yahoo Baba www.yahoobaba.net


File Edit Selection View Go Run Terminal Help
first-project
web.php
routes > web.php
<?php
use Illuminate\Support\Facades\Route;
Route::get('/', function () {
return view('welcome');
});
Route::get('/post/{id?}/comment/{commentid?}', function (string $id = null, string $comment = null) {
if($id){
return "<h1>Post ID : ". $id ."</h1><h2>". $comment ."</h2>";
}else{
return "<h1>No ID Found</h1>";
}
});
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Laravel Routing Parameters & Constraints Tutorial in Hindi / Urdu
web.php
routes > web.php
return view('welcome');
});
Route::get('/post/{id}/comment/{commentid}', function (string $id, string $comment) {
if($id){
return "<h1>Post ID : ". $id ."& Comment : ". $comment ."</h1>";
}else{
return "<h1>No ID Found</h1>";
}
})->where('id', '[0-9]+')->whereAlpha('commentid');
28:15 / 28:36

Laravel Named Routes

http://localhost/page/about

```
Route::get('/page/about-us', function () {
    return 'About Page';
})->name('about');
```

first.blade.php

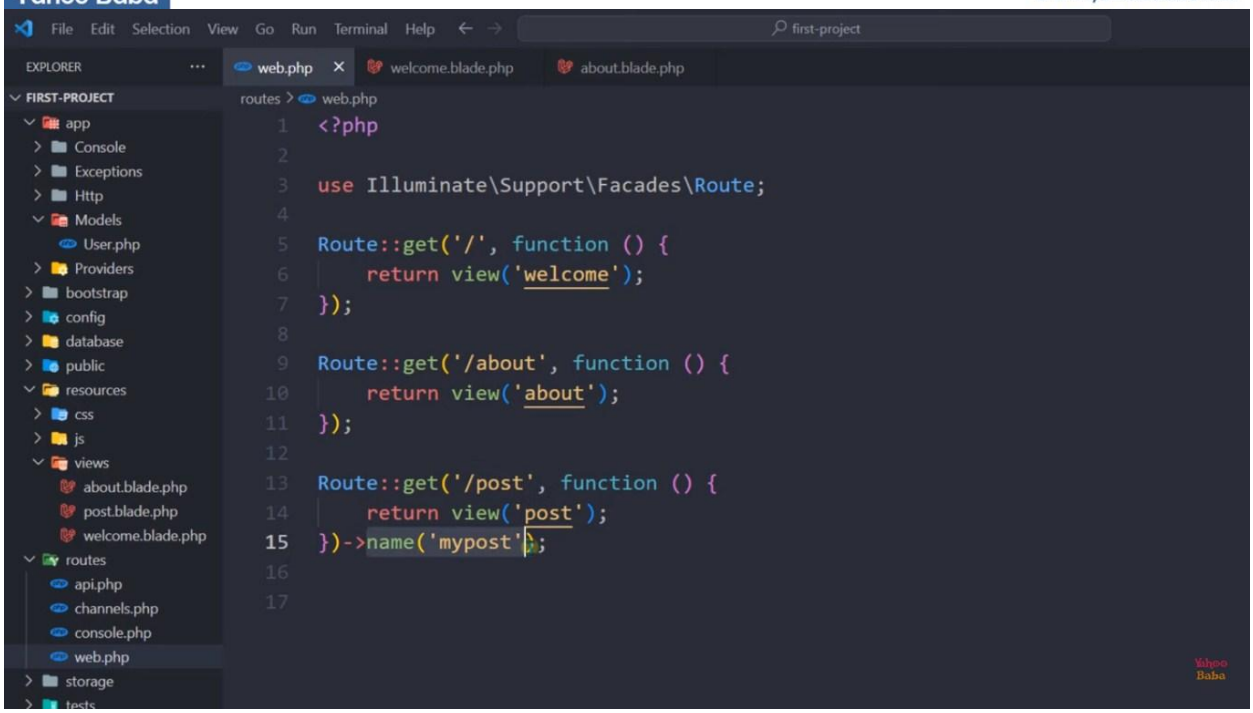
```
<a href='{{ route('about') }}'>About</a>
```

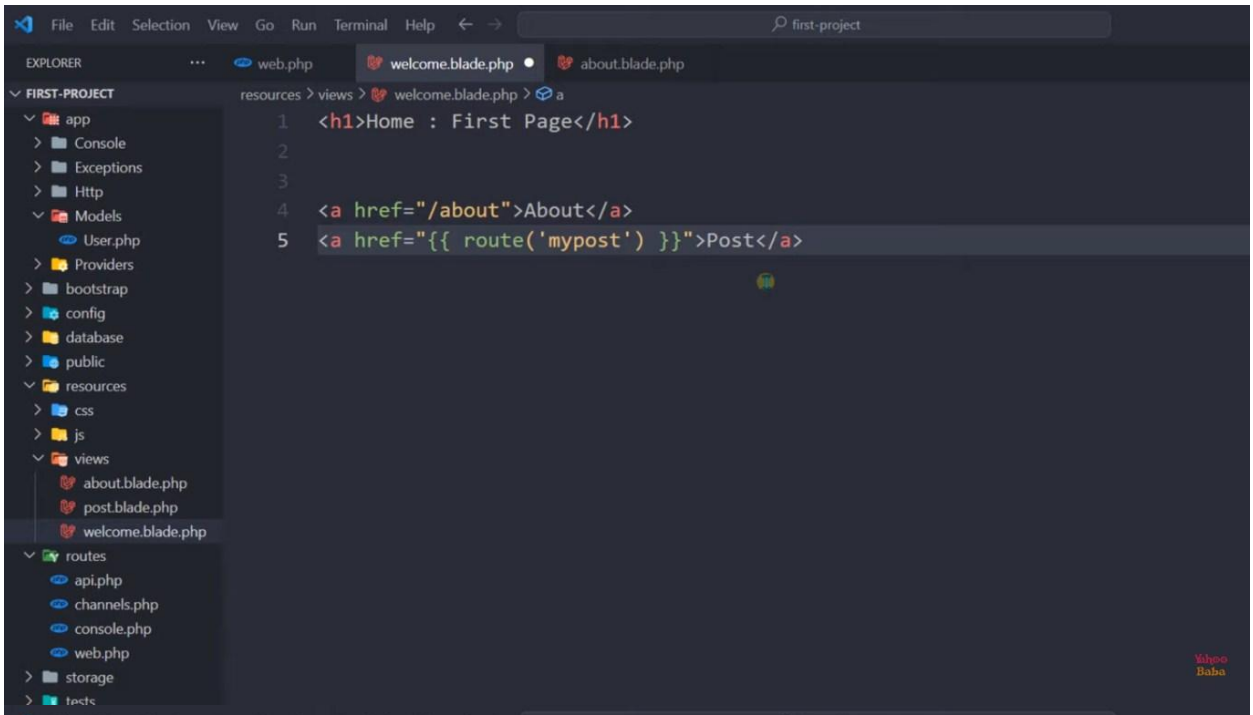
second.blade.php

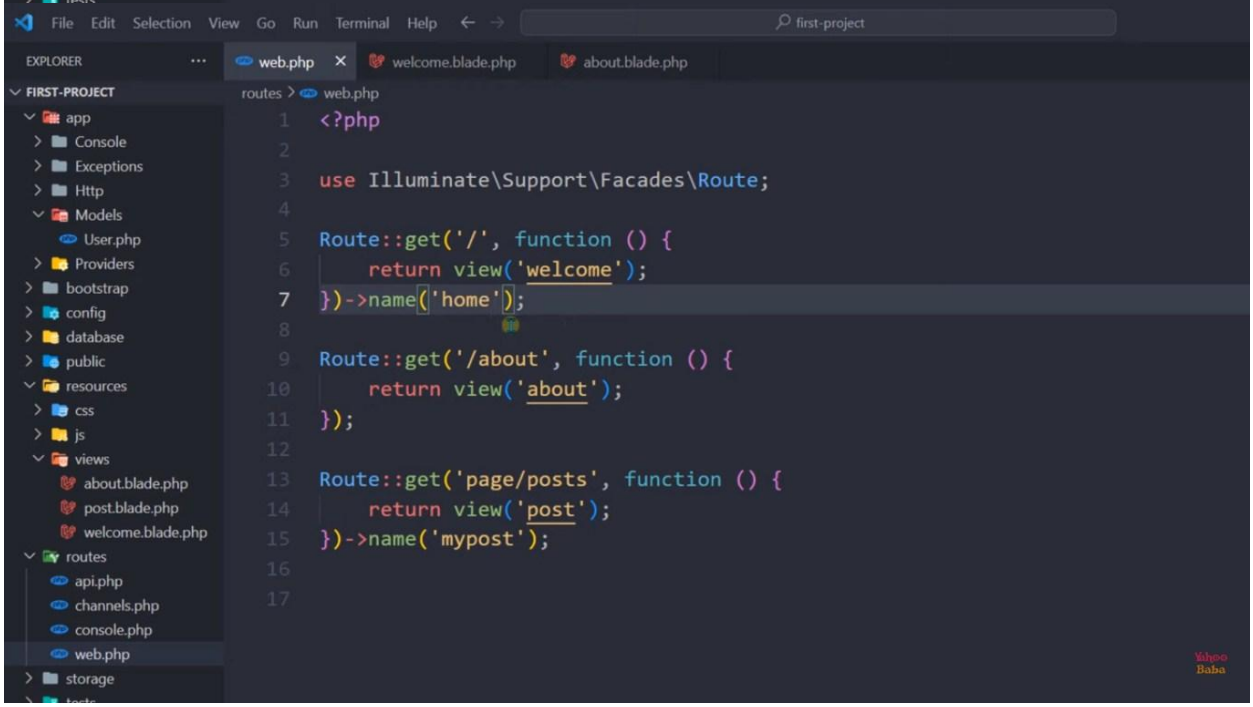
```
<a href='{{ route('about') }}'>About</a>
```

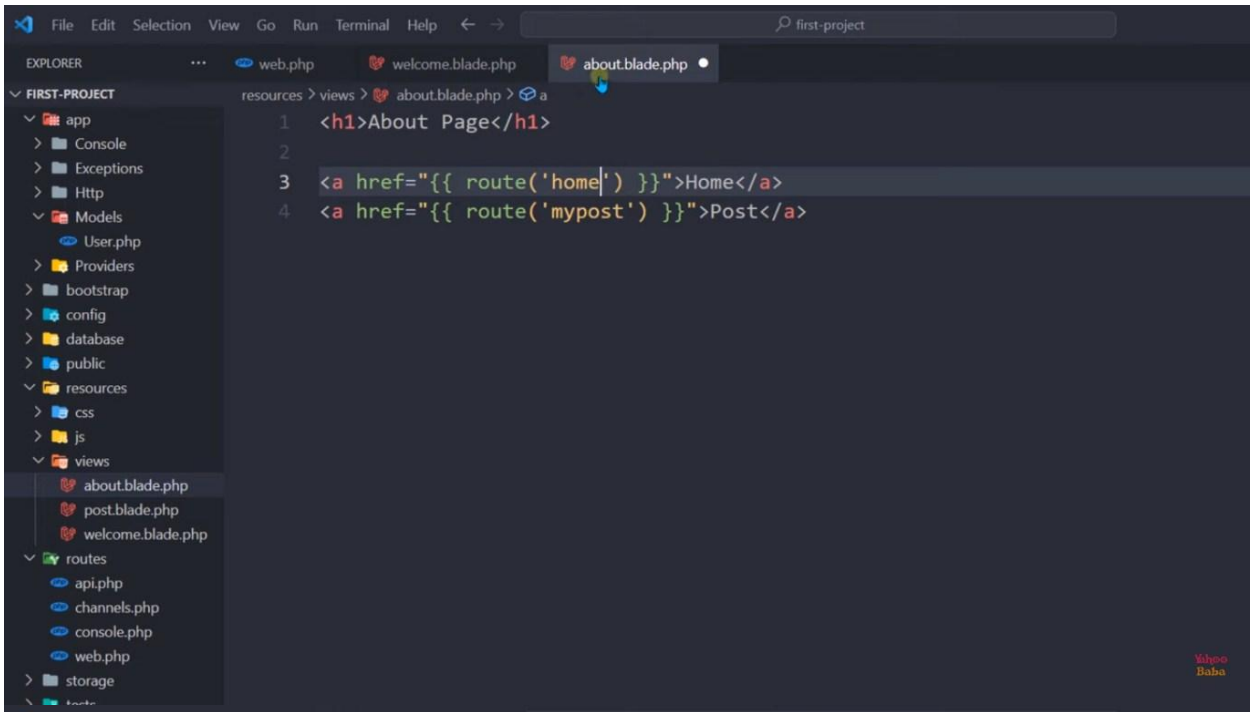
third.blade.php

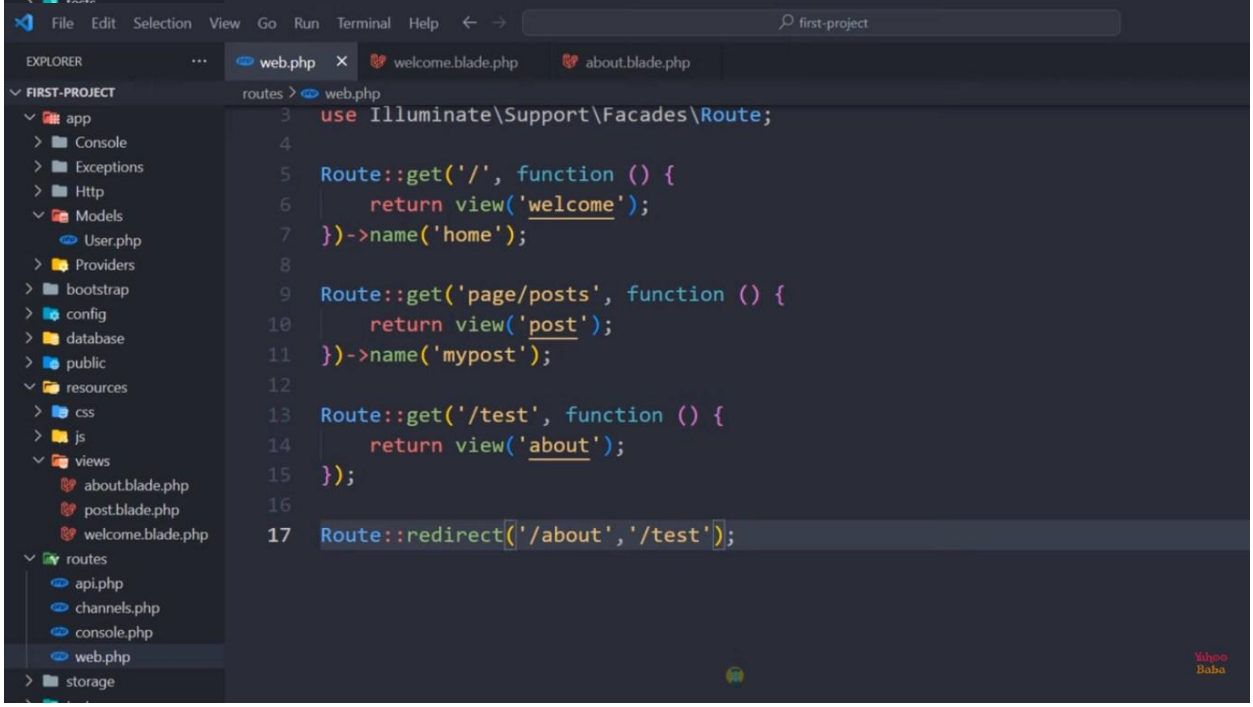
```
<a href='{{ route('about') }}'>About</a>
```


File Edit Selection View Go Run Terminal Help
first-project
EXPLORER
web.php
welcome.blade.php
about.blade.php
FIRST-PROJECT
resources > views > welcome.blade.php > a
app
Console
Exceptions
Http
Models
User.php
Providers
bootstrap
config
database
public
resources
css
js
views
about.blade.php
post.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
1 <h1>Home : First Page</h1>
4 About
5 Post
Yahoo Baba


File Edit Selection View Go Run Terminal Help
first-project
EXPLORER
web.php
welcome.blade.php
about.blade.php
FIRST-PROJECT
routes > web.php
app
Console
Exceptions
Http
Models
User.php
Providers
bootstrap
config
database
public
resources
css
js
views
about.blade.php
post.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
1 <?php
3 use Illuminate\Support\Facades\Route;
5 Route::get('/', function () {
6 return view('welcome');
7 })->name('home');
9 Route::get('/about', function () {
10 return view('about');
11 });
13 Route::get('page/posts', function () {
14 return view('post');
15 })->name('mypost');
Yahoo Baba

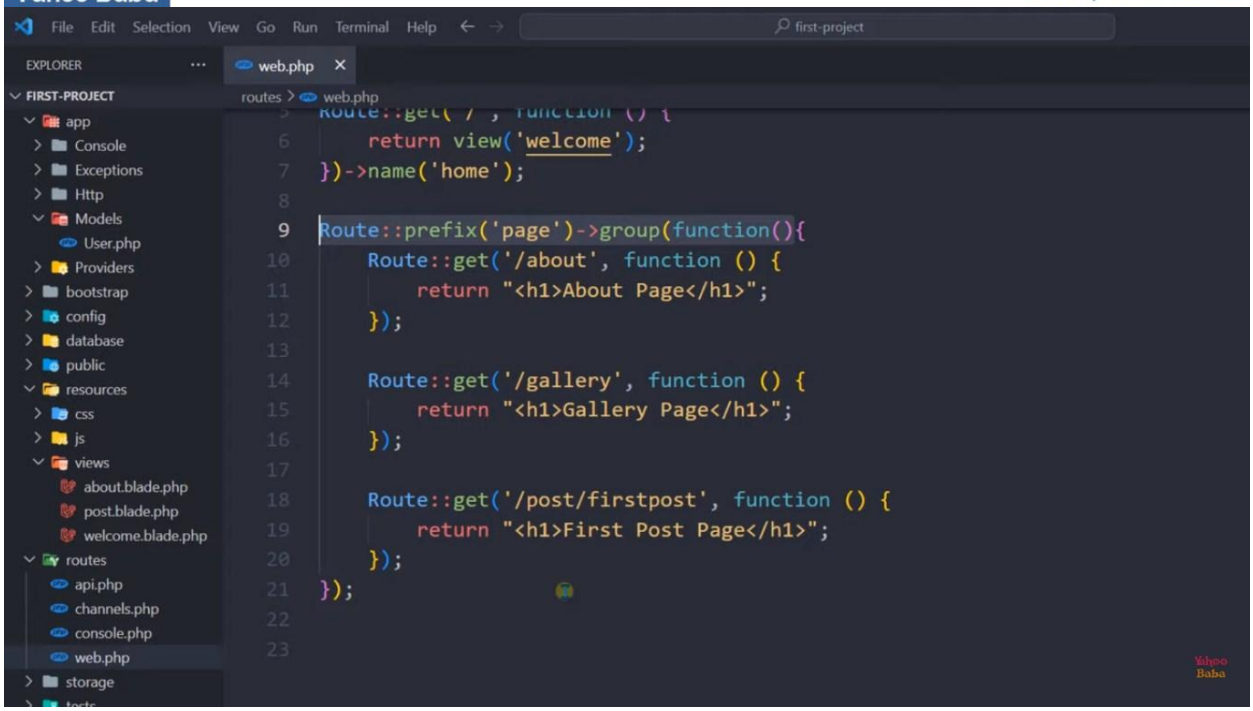

File Edit Selection View Go Run Terminal Help
first-project
EXPLORER
web.php
welcome.blade.php
about.blade.php
FIRST-PROJECT
resources > views > about.blade.php > a
app
Console
Exceptions
Http
Models
User.php
Providers
bootstrap
config
database
public
resources
css
js
views
about.blade.php
post.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
1 <h1>About Page</h1>
2
3 Home
4 Post
Yahoo Baba

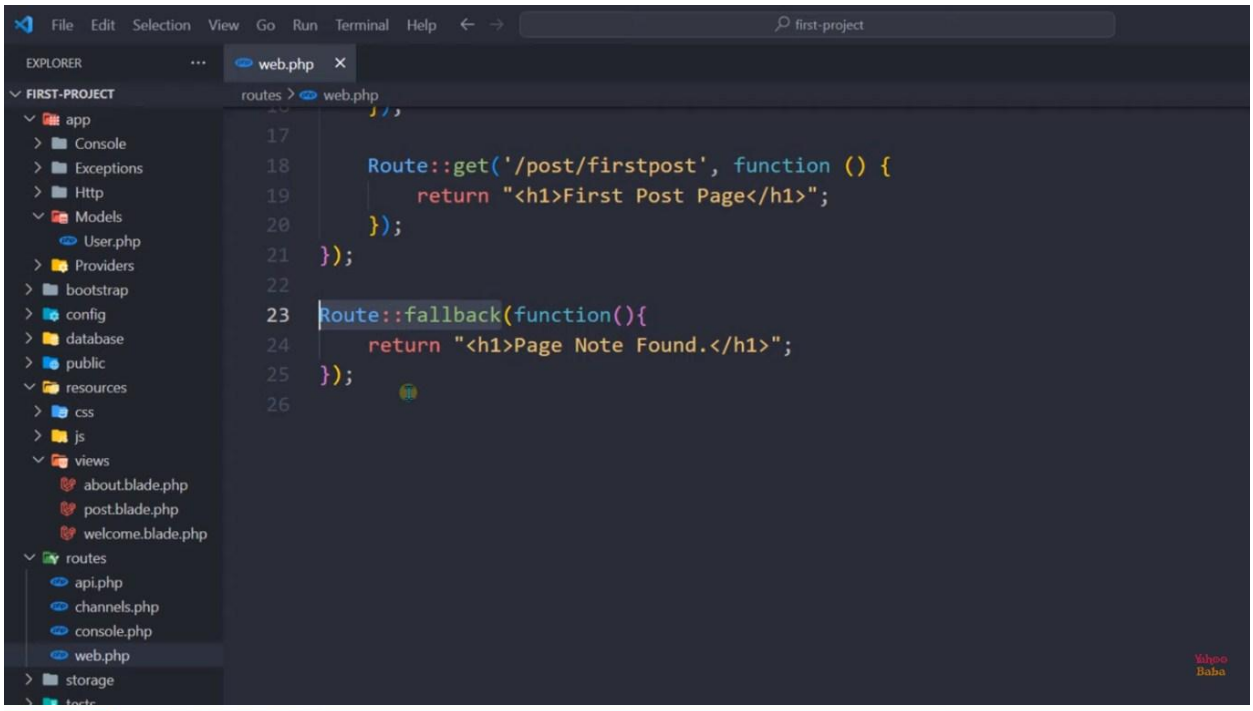

File Edit Selection View Go Run Terminal Help
first-project
EXPLORER
web.php
welcome.blade.php
about.blade.php
FIRST-PROJECT
routes > web.php
app
Console
Exceptions
Http
Models
User.php
Providers
bootstrap
config
database
public
resources
css
js
views
about.blade.php
post.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6 return view('welcome');
7 })->name('home');
8
9 Route::get('page/posts', function () {
10 return view('post');
11 })->name('mypost');
12
13 Route::get('/test', function () {
14 return view('about');
15 });
16
17 Route::redirect('/about','/test');
Yahoo Baba

Laravel Route Groups

```
Route::prefix('page')->group(function () {
    Route::get('/post/1', function () {
    });
    Route::get('/about/', function () {
    });
    Route::get('/gallery/', function () {
    });
});
```

→ http://localhost/page/about





What is Blade Template ?

Blade Template

↓

Template Engine Based on PHP

Blade provides a clean and convenient way to create views in Laravel

Benefits : Create Dynamic and Reusable Templates

HTML & PHP

Laravel : Blade Template Syntax

```
<?php
   echo "Hello";
?>
```

```
{{ "Hello" }}
```

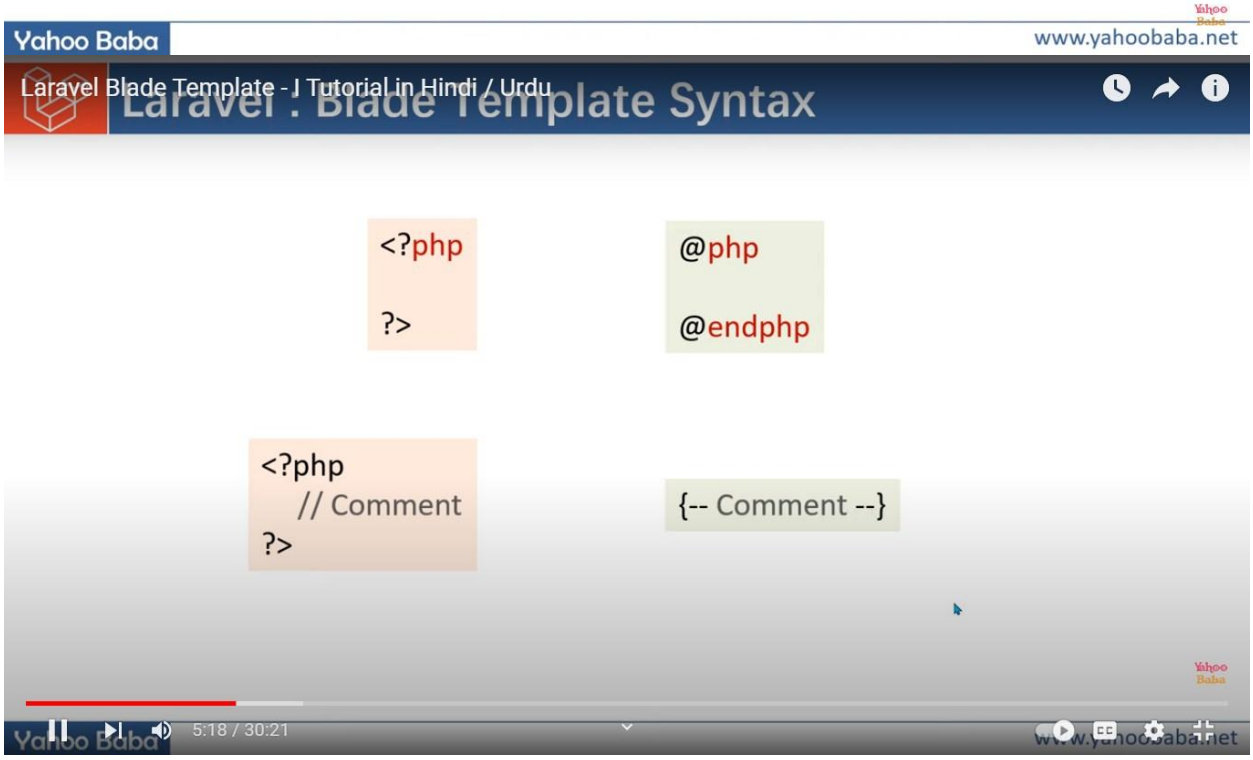
Prevent cross-site scripting (XSS) Attacks

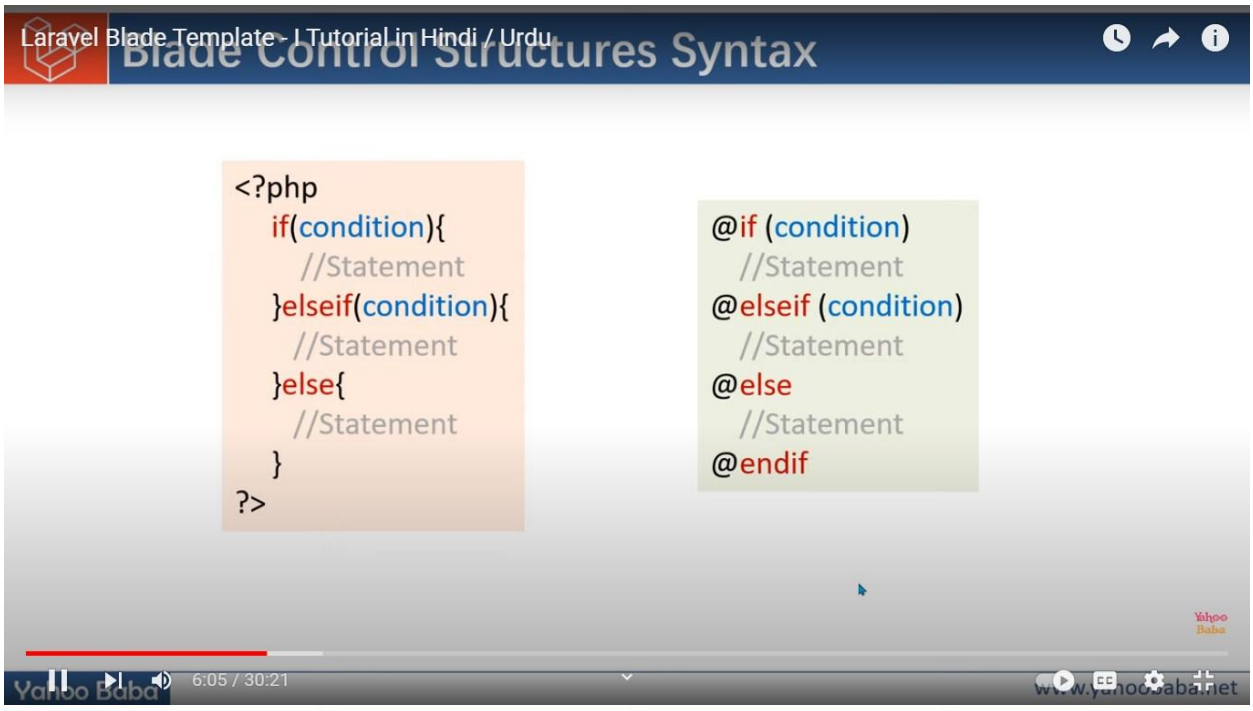
```
<?php
   echo $name;
?>
```

```
{{ $name }}
```

```
<?php
   echo "<h1>Hello</h1>";
?>
```

```
{!! "<h1>Hello</h1>" !!}
```





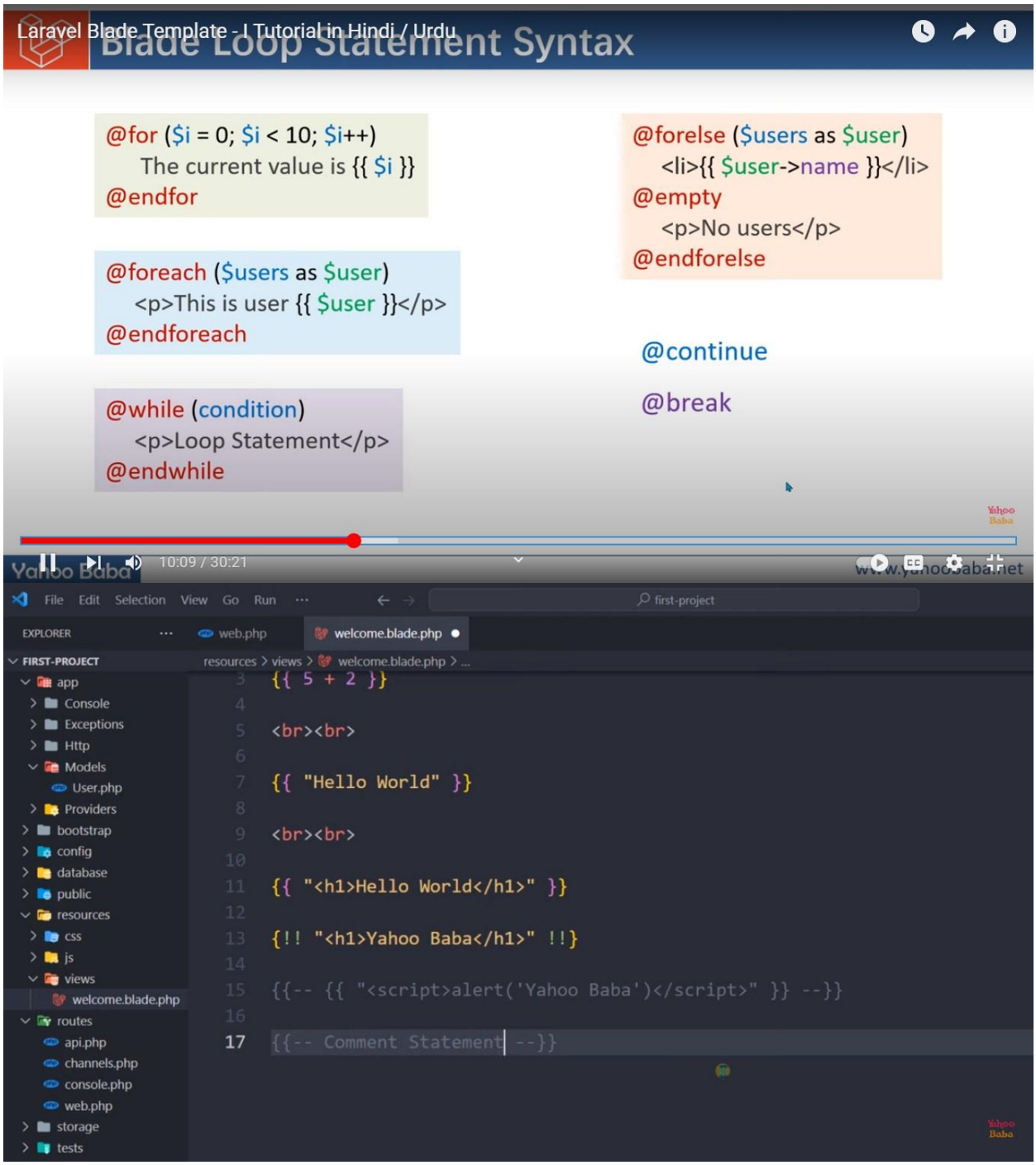
Blade Control Structures Syntax

```
<?php
    if(isset($records)){
        //Statement
    }
?>
```

```
@isset($records)
    // $records is defined and is not null...
@endisset
```

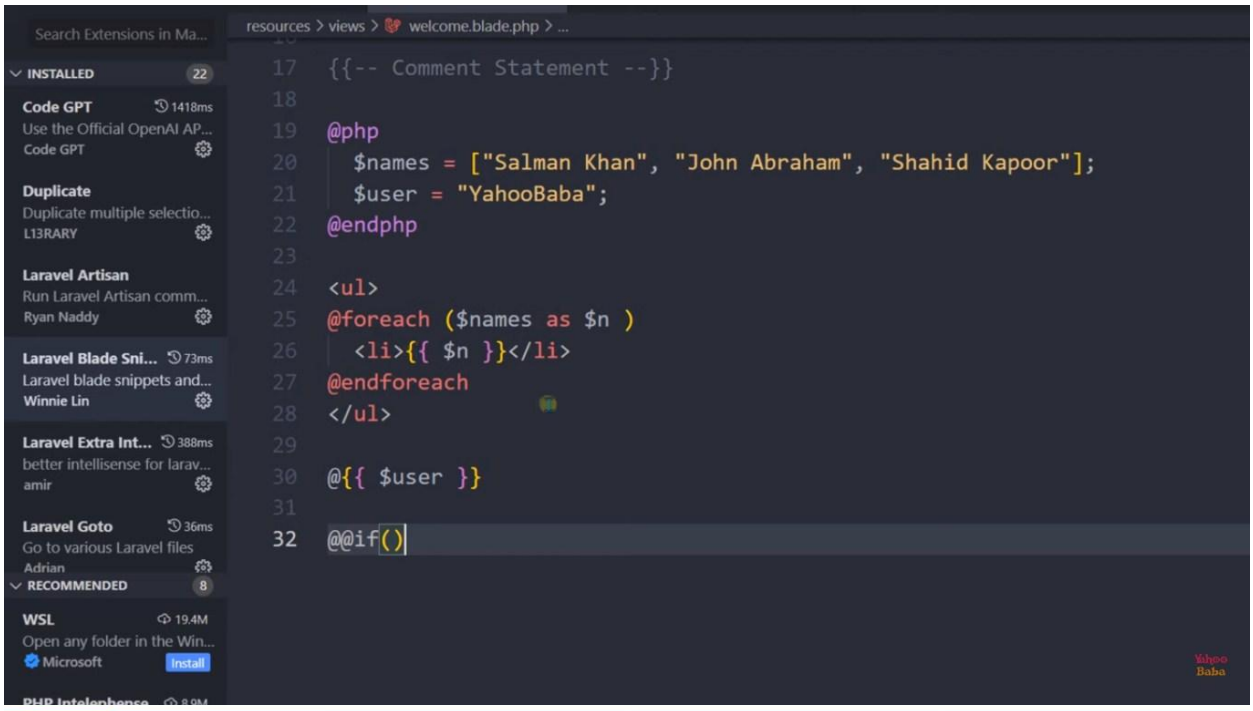
```
<?php
    if(emtpy($records)){
        //Statement
    }
?>
```

```
@empty($records)
    // $records is "empty"...
@endempty
```


Laravel Blade Template - I Tutorial in Hindi / Urdu
Blade Loop Statement Syntax
@for ($i = 0; $i < 10; $i++)
The current value is {{ $i }}
@endfor
@foreach ($users as $user)
This is user {{ $user }}
@endforeach
@while (condition)
Loop Statement
@endwhile
@forelse ($users as $user)
{{ $user->name }}
@empty
No users
@endforelse
@continue
@break
10:09 / 30:21
File Edit Selection View Go Run
first-project
EXPLORER
web.php
welcome.blade.php
FIRST-PROJECT
resources > views > welcome.blade.php > ...
app
Console
Exceptions
Http
Models
User.php
Providers
bootstrap
config
database
public
resources
css
js
views
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
{{ 5 + 2 }}

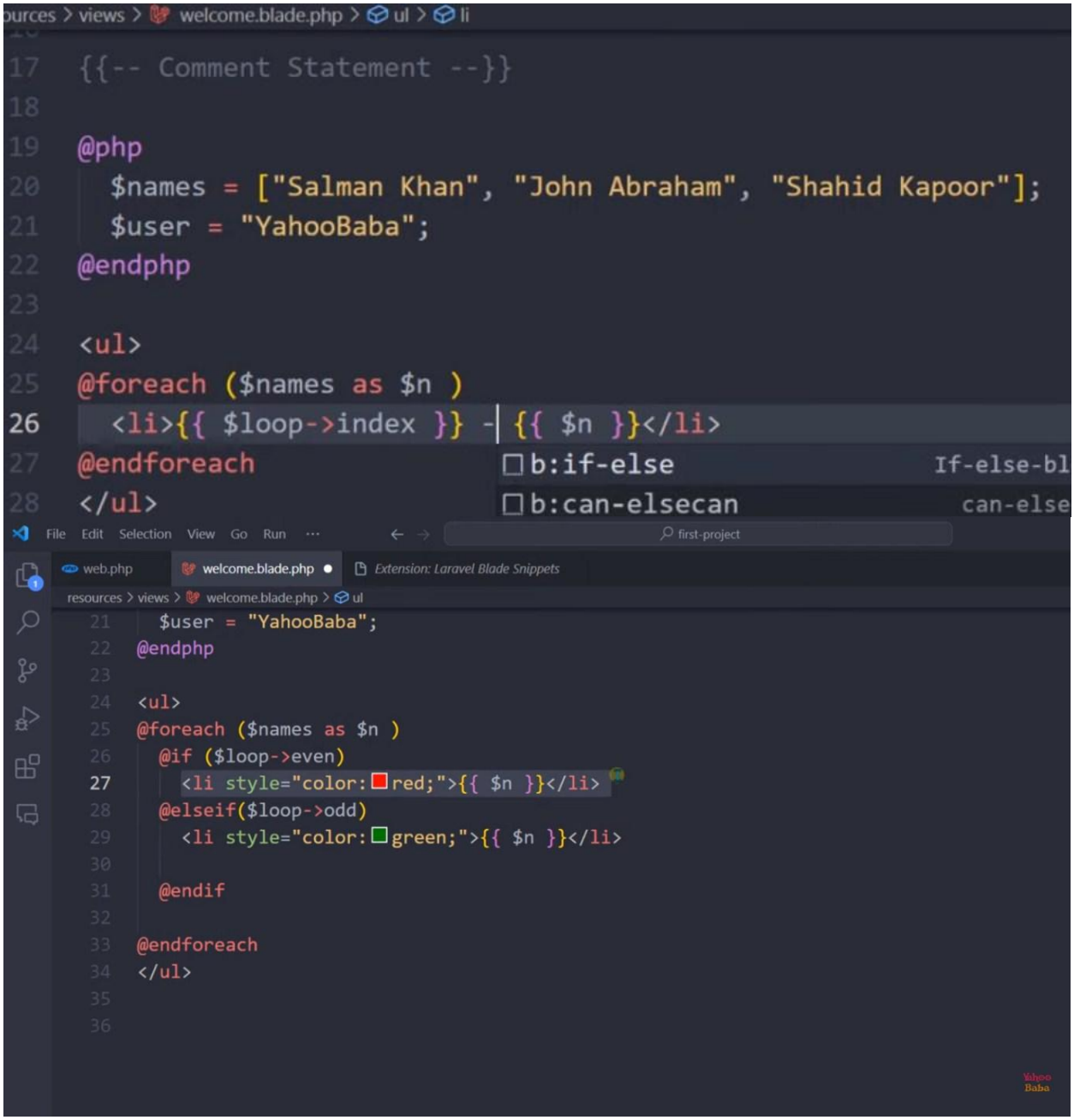
{{ "Hello World" }}

{{ "<h1>Hello World</h1>" }}
{!! "<h1>Yahoo Baba</h1>" !!}
{{-- {{ "<script>alert('Yahoo Baba')</script>" }} --}}
{{-- Comment Statement --}}



Blade Loop Variable for @foreach

Property	Description
$loop->index	The index of the current loop iteration (starts at 0).
$loop->iteration	The current loop iteration (starts at 1).
$loop->remaining	The iterations remaining in the loop.
$loop->count	The total number of items in the array being iterated.
$loop->first	Whether this is the first iteration through the loop.
$loop->last	Whether this is the last iteration through the loop.
$loop->even	Whether this is an even iteration through the loop.
$loop->odd	Whether this is an odd iteration through the loop.
$loop->depth	The nesting level of the current loop.
$loop->parent	When in a nested loop, the parent's loop variable.


ources > views > welcome.blade.php > ul > li
{{-- Comment Statement --}}
@php
$names = ["Salman Khan", "John Abraham", "Shahid Kapoor"];
$user = "YahooBaba";
@endphp

@foreach ($names as $n)
{{ $loop->index }} - {{ $n }}
@endforeach

b:if-else
If-else-bl
b:can-elsecan
can-else
File Edit Selection View Go Run ...
first-project
web.php
welcome.blade.php
Extension: Laravel Blade Snippets
resources > views > welcome.blade.php > ul
$user = "YahooBaba";
@endphp

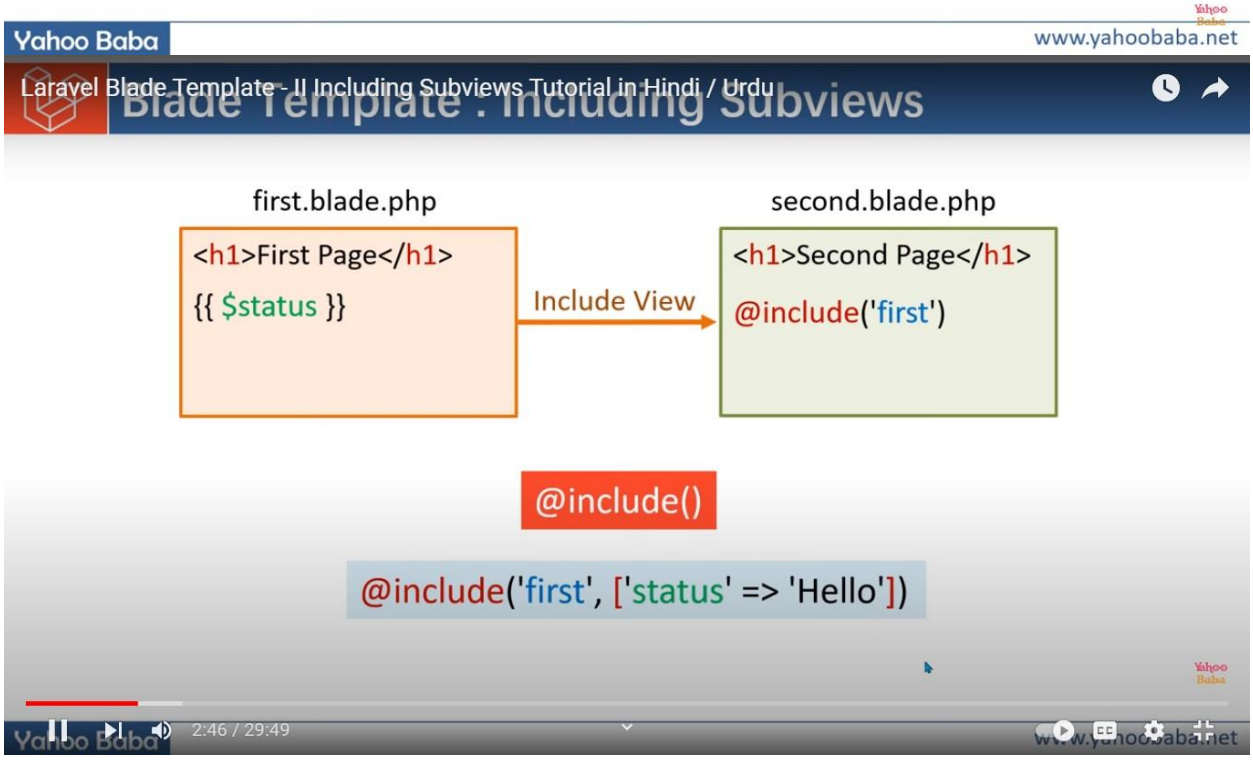
@foreach ($names as $n)
@if ($loop->even)
<li style="color: red;">{{ $n }}
@elseif($loop->odd)
<li style="color: green;">{{ $n }}
@endif
@endforeach

Yahoo Baba

Blade Template Main Directives

- @include
- @section
- @extend
- @yield

Reusable Templates



Blade Template : Including Subviews

first.blade.php

```
<h1>First Page</h1>
{{ $status }}
```

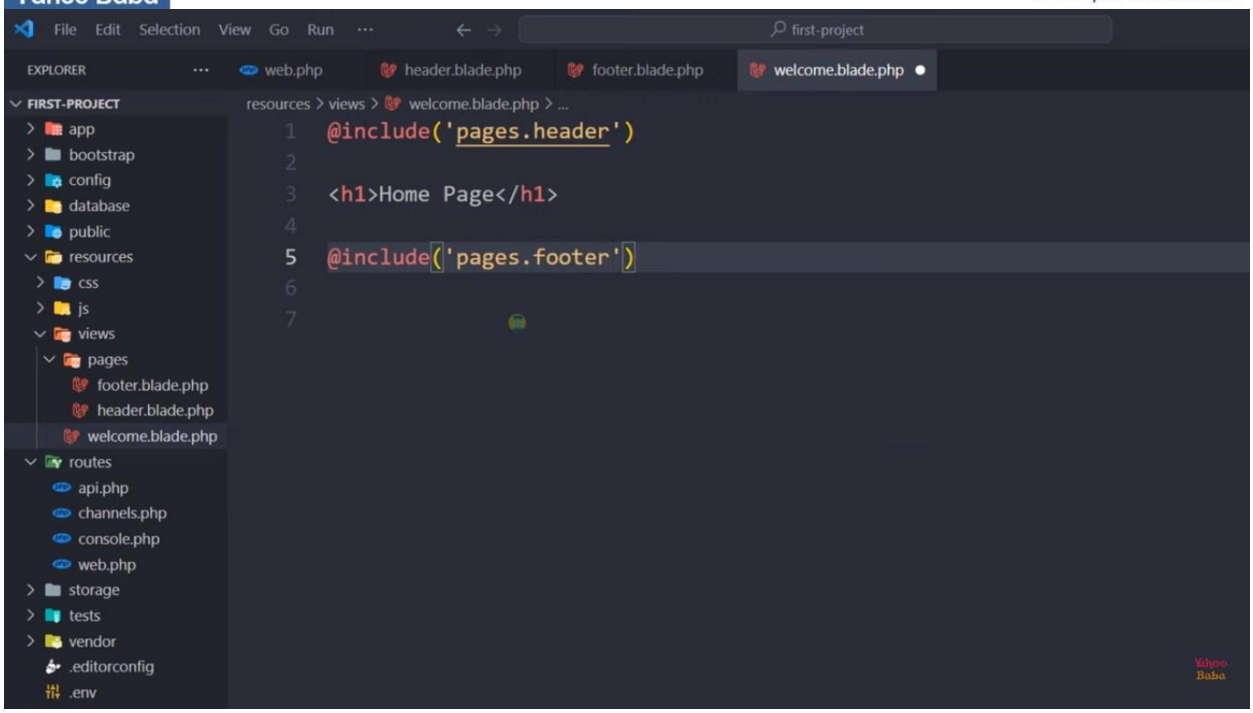
Include View

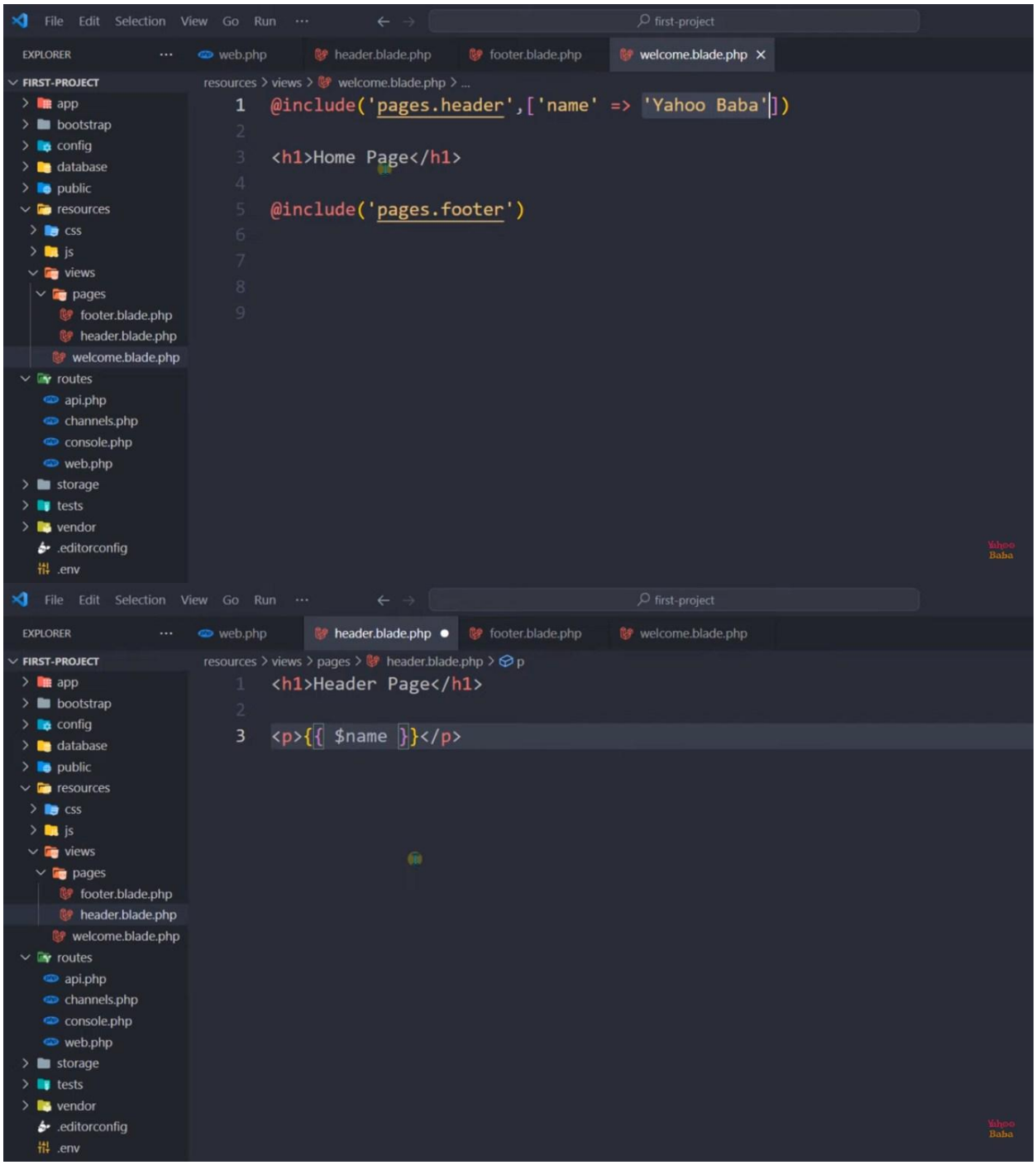
second.blade.php

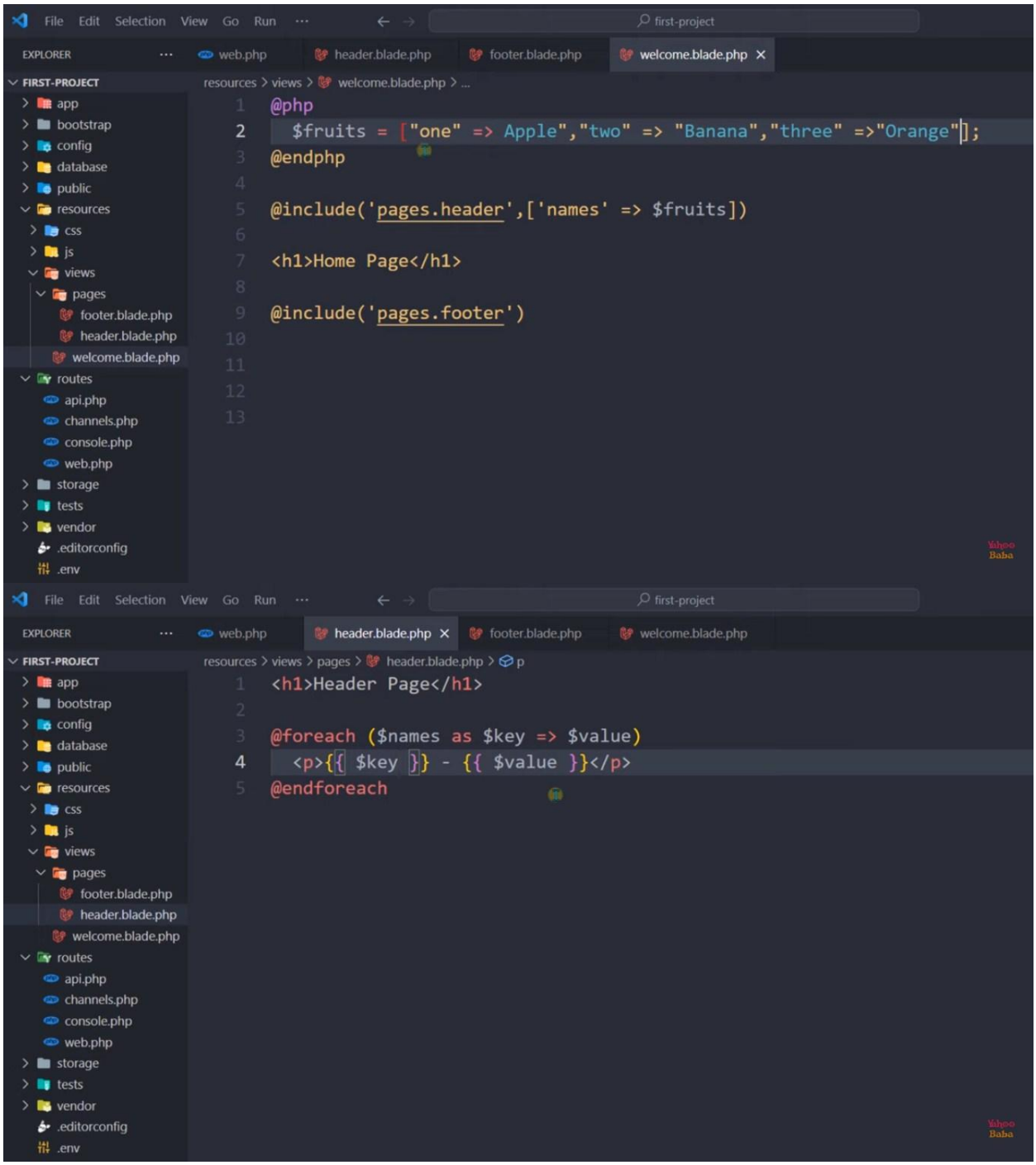
```
<h1>Second Page</h1>
@include('first')
```

@include()

```
@include('first', ['status' => 'Hello'])
```


File Edit Selection View Go Run
first-project
EXPLORER
web.php
header.blade.php
footer.blade.php
welcome.blade.php
FIRST-PROJECT
app
bootstrap
config
database
public
resources
css
js
views
pages
footer.blade.php
header.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
.editorconfig
.env
resources > views > welcome.blade.php > ...
1 @include('pages.header',['name' => 'Yahoo Baba'])
3 <h1>Home Page</h1>
5 @include('pages.footer')
resources > views > pages > header.blade.php > p
1 <h1>Header Page</h1>
3 {{ $name }}
Yahoo Baba

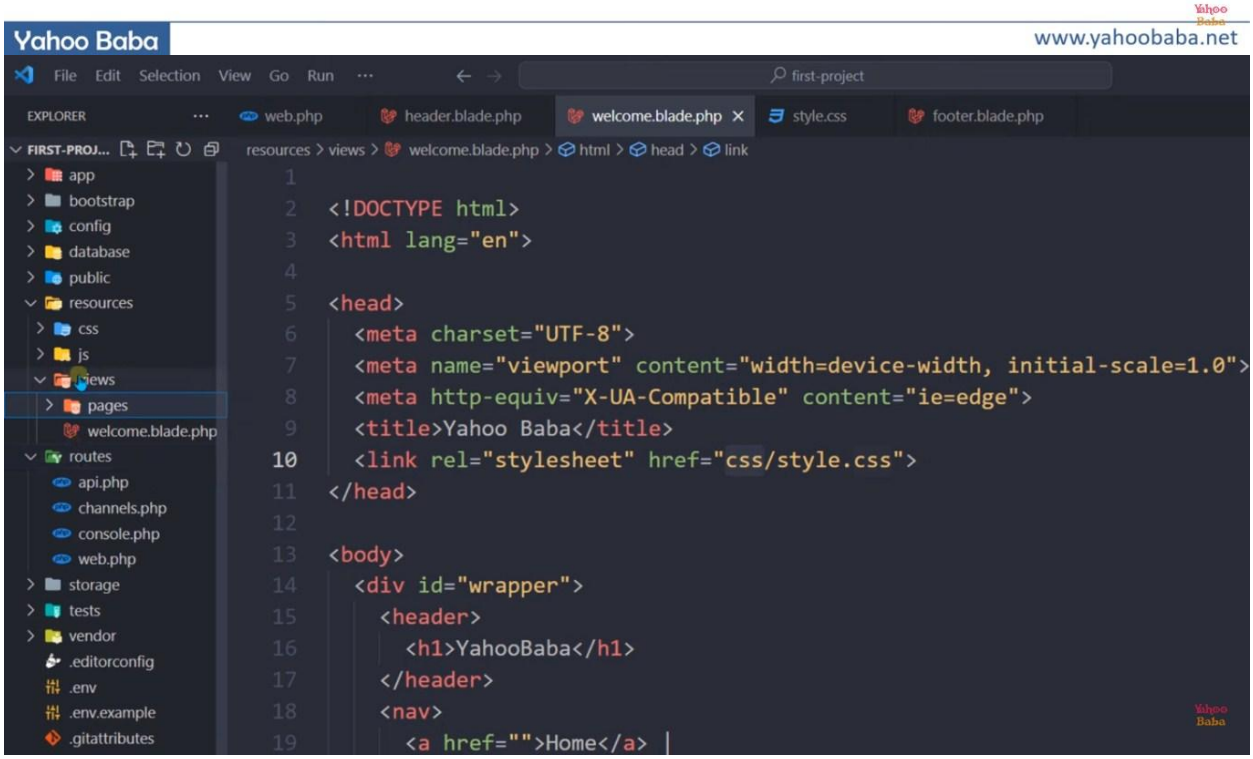

File Edit Selection View Go Run
first-project
EXPLORER
web.php
header.blade.php
footer.blade.php
welcome.blade.php
FIRST-PROJECT
resources > views > welcome.blade.php > ...
app
bootstrap
config
database
public
resources
css
js
views
pages
footer.blade.php
header.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
.editorconfig
.env
@php
$fruits = ["one" => Apple","two" => "Banana","three" =>"Orange"];
@endphp
@include('pages.header',['names' => $fruits])
<h1>Home Page</h1>
@include('pages.footer')
Yahoo Baba
File Edit Selection View Go Run
first-project
EXPLORER
web.php
header.blade.php
footer.blade.php
welcome.blade.php
FIRST-PROJECT
resources > views > pages > header.blade.php > p
app
bootstrap
config
database
public
resources
css
js
views
pages
footer.blade.php
header.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
.editorconfig
.env
<h1>Header Page</h1>
@foreach ($names as $key => $value)
{{ $key }} - {{ $value }}
@endforeach
Yahoo Baba

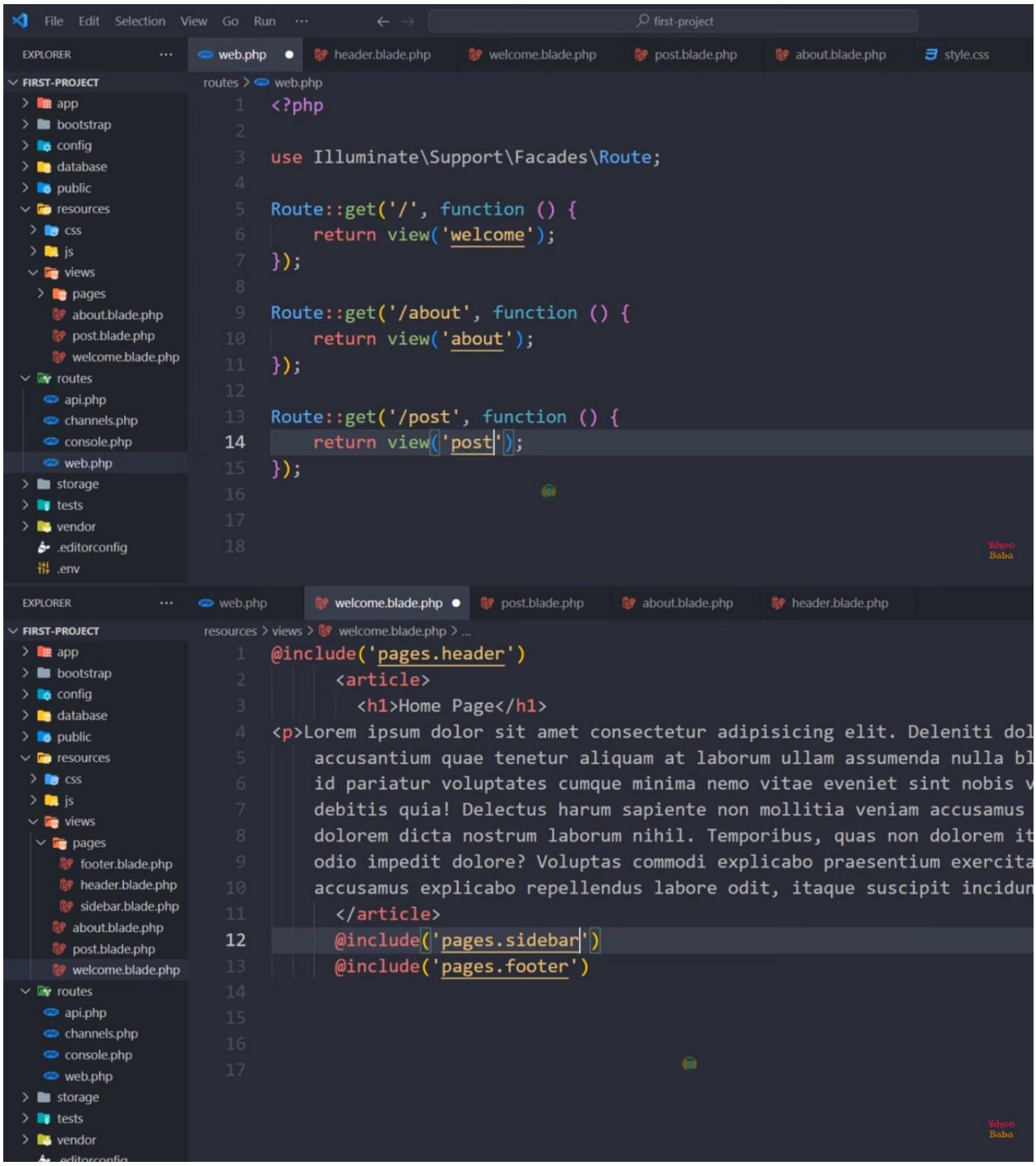
Including Subviews with Conditional Check

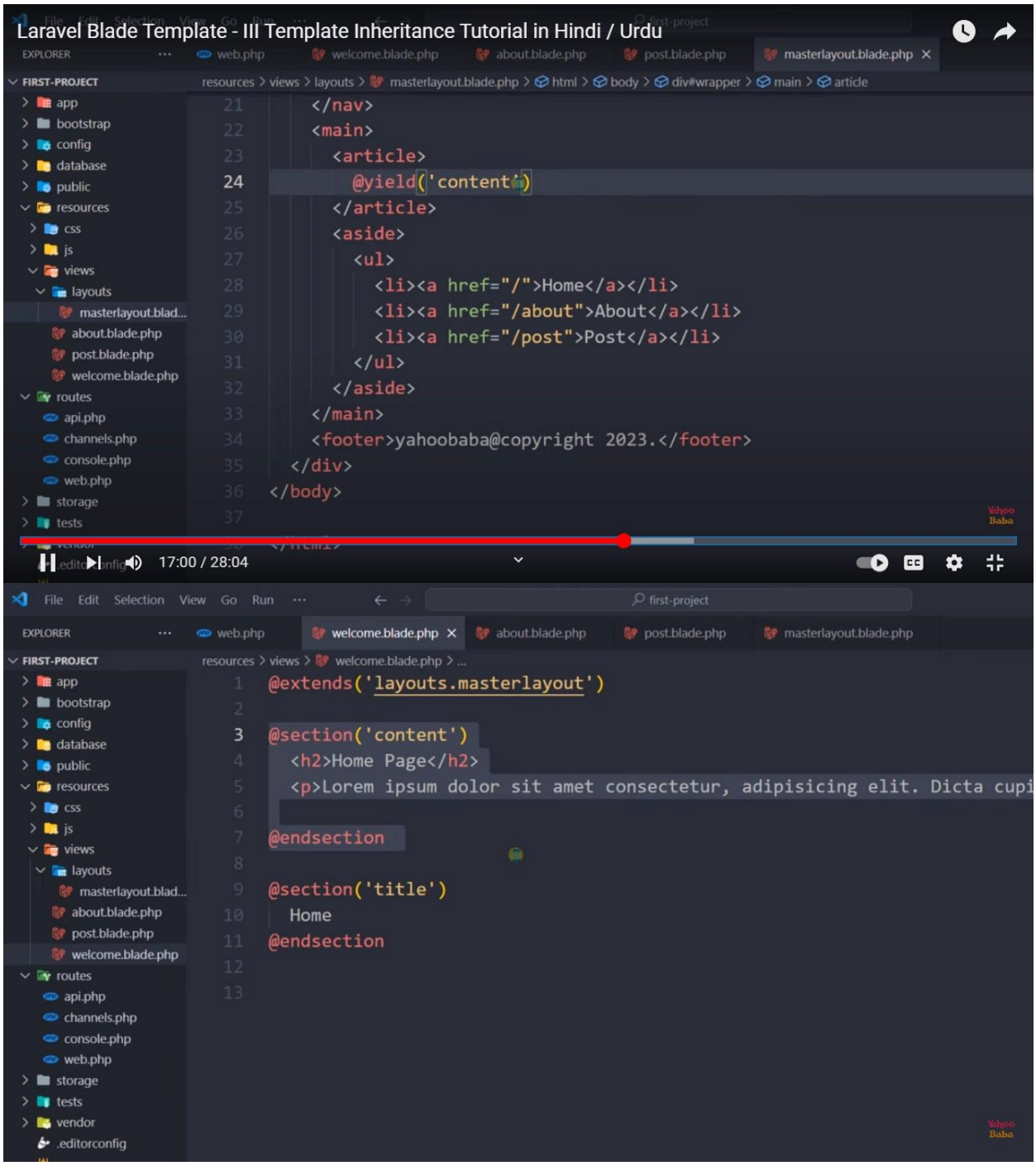
```
@includeWhen (Condition Value,'viewfile', ['status' => 'Hello'])
```

TRUE / FALSE

```
@includeUnless (Condition Value,'viewfile', ['status' => 'Hello'])
```



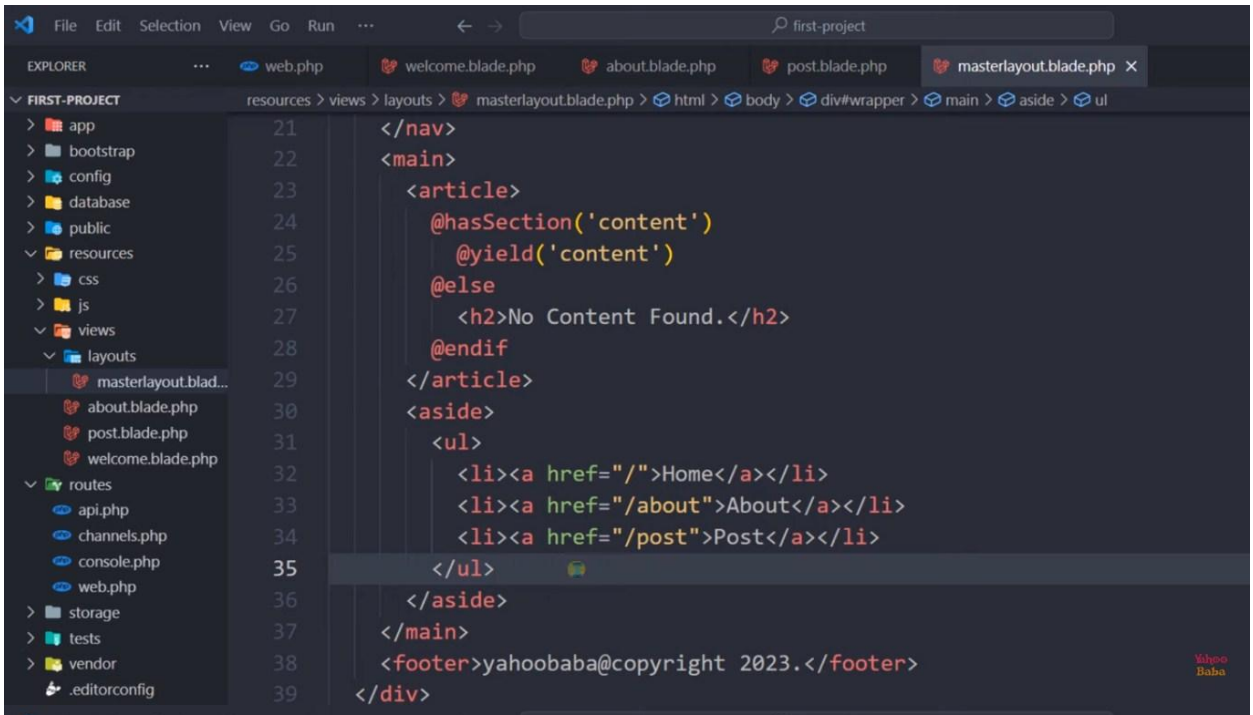

File Edit Selection View Go Run
first-project
EXPLORER
web.php
header.blade.php
welcome.blade.php
post.blade.php
about.blade.php
style.css
FIRST-PROJECT
routes > web.php
app
bootstrap
config
database
public
resources
css
js
views
pages
about.blade.php
post.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
.editorconfig
.env
<?php
use Illuminate\Support\Facades\Route;
Route::get('/', function () {
return view('welcome');
});
Route::get('/about', function () {
return view('about');
});
Route::get('/post', function () {
return view('post');
});
EXPLORER
web.php
welcome.blade.php
post.blade.php
about.blade.php
header.blade.php
FIRST-PROJECT
resources > views > welcome.blade.php > ...
app
bootstrap
config
database
public
resources
css
js
views
pages
footer.blade.php
header.blade.php
sidebar.blade.php
about.blade.php
post.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
@include('pages.header')
<article>
<h1>Home Page</h1>
<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Deleniti dol
accusantium quae tenetur aliquam at laborum ullam assumenda nulla bl
id pariatur voluptates cumque minima nemo vitae eveniet sint nobis v
debitis quia! Delectus harum sapiente non mollitia veniam accusamus
dolorem dicta nostrum laborum nihil. Temporibus, quas non dolorem it
odio impedit dolore? Voluptas commodi explicabo praesentium exercita
accusamus explicabo repellendus labore odit, itaque suscipit incidun
</article>
@include('pages.sidebar')
@include('pages.footer')


Laravel Blade Template - III Template Inheritance Tutorial in Hindi / Urdu
EXPLORER
web.php
welcome.blade.php
about.blade.php
post.blade.php
masterlayout.blade.php
FIRST-PROJECT
resources > views > layouts > masterlayout.blade.php > html > body > div#wrapper > main > article
app
bootstrap
config
database
public
resources
css
js
views
layouts
masterlayout.blad...
about.blade.php
post.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
.editorconfig
</nav>
<main>
<article>
@yield('content')
</article>
<aside>

Home
About
Post

</aside>
</main>
<footer>yahoobaba@copyright 2023.</footer>

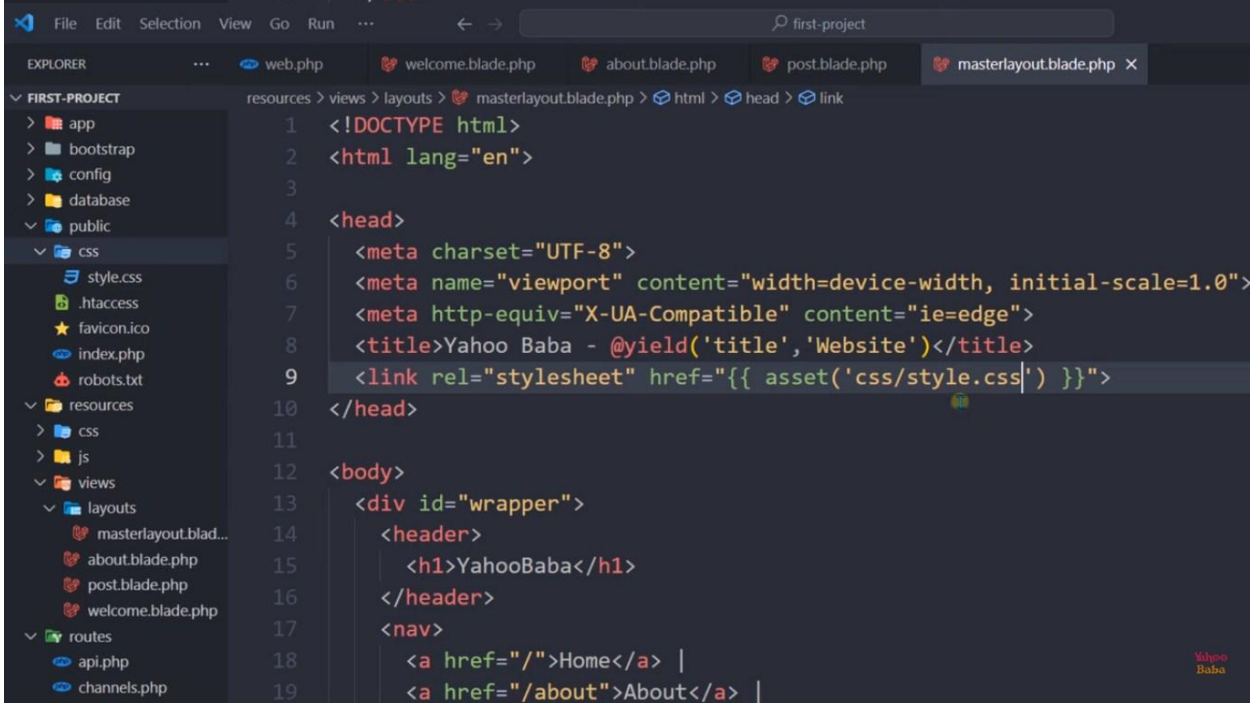
</body>
17:00 / 28:04
File Edit Selection View Go Run
first-project
resources > views > welcome.blade.php > ...
@extends('layouts.masterlayout')
@section('content')
<h2>Home Page</h2>
<p>Lorem ipsum dolor sit amet consectetur, adipisicing elit. Dicta cupi
@endsection
@section('title')
Home
@endsection


File Edit Selection View Go Run
first-project
EXPLORER
web.php
welcome.blade.php
about.blade.php
post.blade.php
masterlayout.blade.php
FIRST-PROJECT
resources > views > layouts > masterlayout.blade.php > html > body > div#wrapper > main > aside > ul
app
bootstrap
config
database
public
resources
css
js
views
layouts
masterlayout.blad...
about.blade.php
post.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
.editorconfig
</nav>
<main>
<article>
@hasSection('content')
@yield('content')
@else
<h2>No Content Found.</h2>
@endif
</article>
<aside>

Home
About
Post

</aside>
</main>
<footer>yahoobaba@copyright 2023.</footer>

Yahoo Baba


File Edit Selection View Go Run
first-project
EXPLORER
web.php
welcome.blade.php
about.blade.php
post.blade.php
masterlayout.blade.php
FIRST-PROJECT
resources > views > layouts > masterlayout.blade.php > html > head > link
app
bootstrap
config
database
public
css
style.css
.htaccess
favicon.ico
index.php
robots.txt
resources
css
js
views
layouts
masterlayout.blad...
about.blade.php
post.blade.php
welcome.blade.php
routes
api.php
channels.php
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Yahoo Baba - @yield('title','Website')</title>
<link rel="stylesheet" href="{{ asset('css/style.css') }}">
</head>
<body>

<header>
<h1>YahooBaba</h1>
</header>
<nav>
Home |
About |
Yahoo Baba

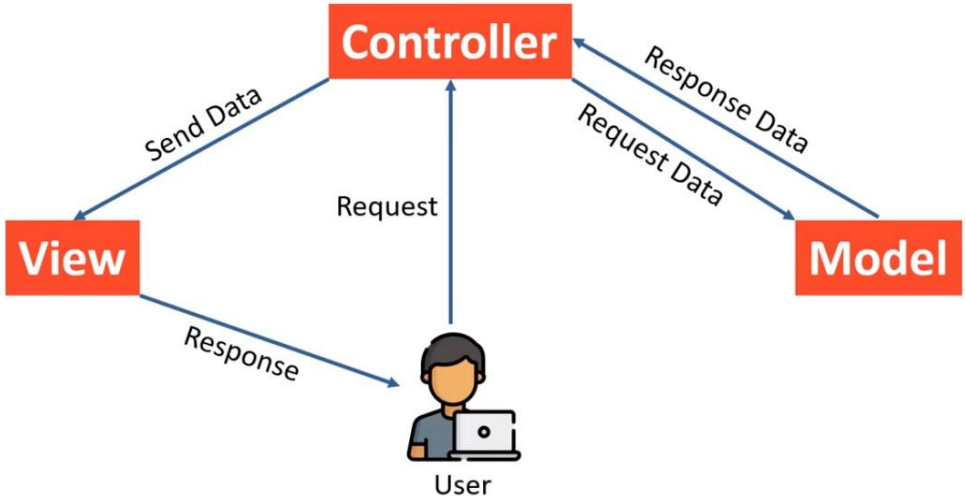
Passing Data : Route to View


```
Route::get('/users', function () {
   $name = 'Yahoo Baba';
   return view('userpage', ['user' => $name] );
});
```

```
<h1>{{ $user }}</h1>
```

userpage.blade.php

Controller Role In MVC Framework



Laravel : Controller

1. Create Controller File

```
php artisan make:controller UserController
```

2. Create Controller Class

```
class UserController extends Controller
{
    public function show()
    {
        return view('user.profile');
    }
}
```

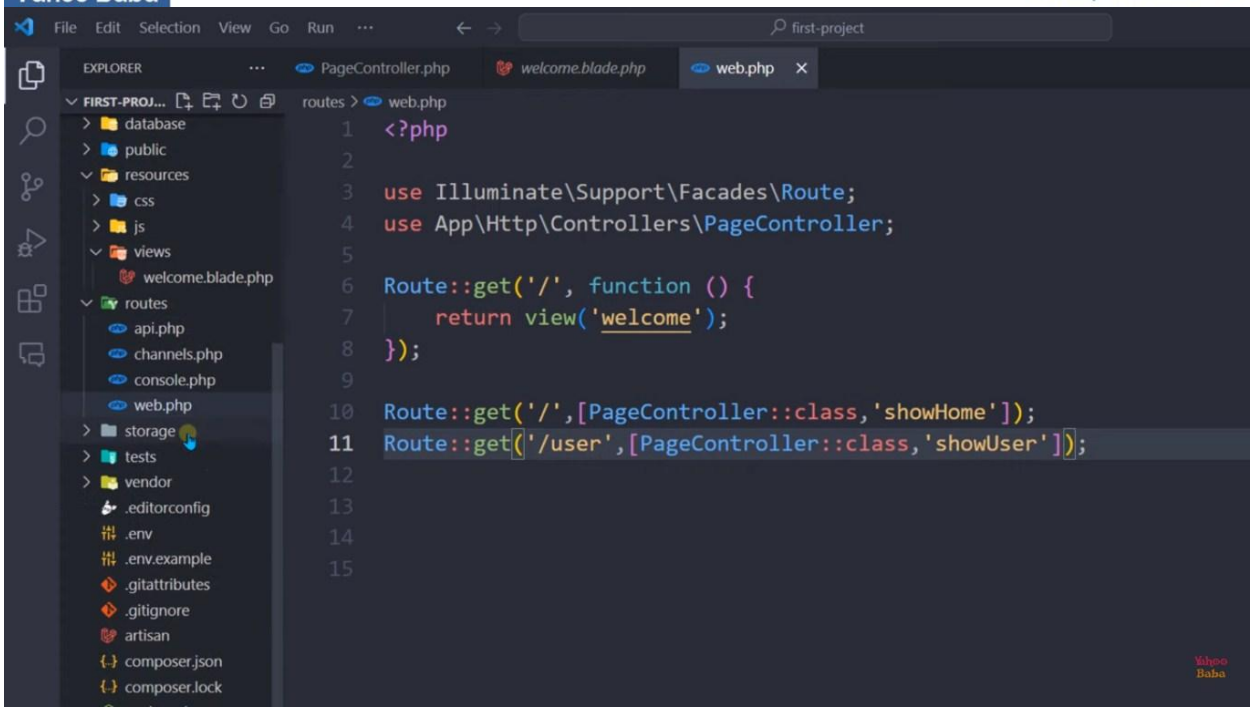
UserController.php

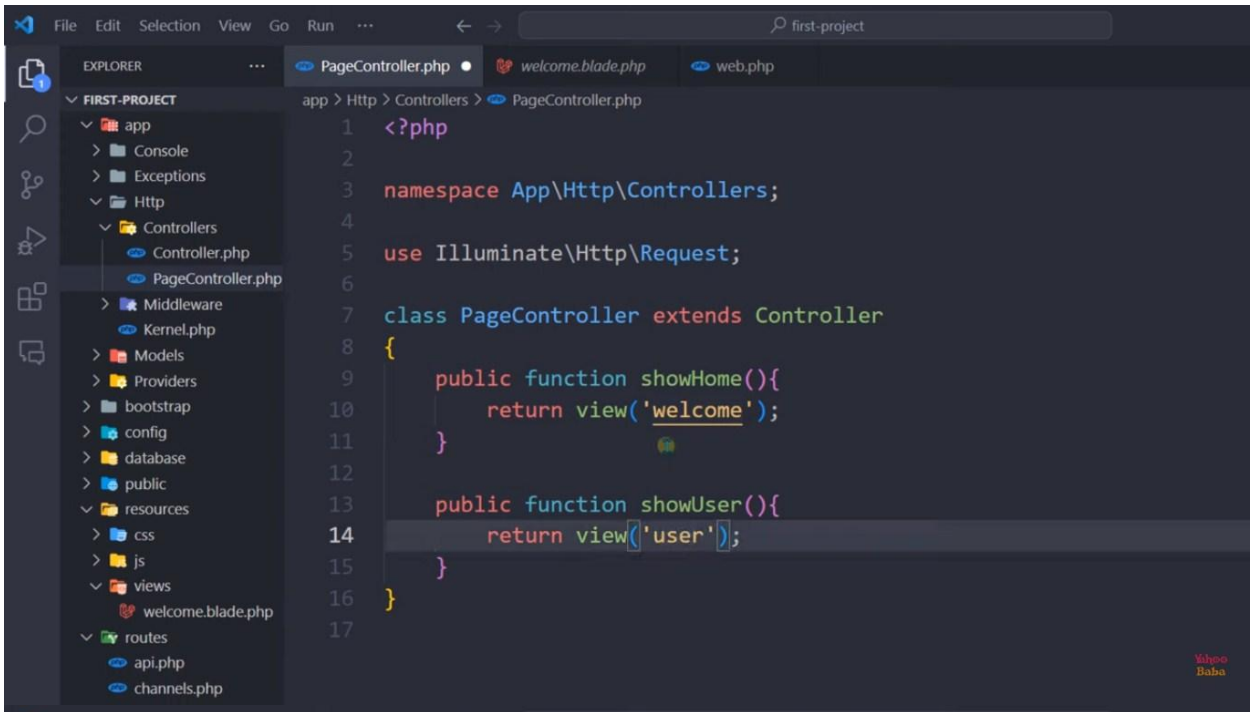
3. Create Route

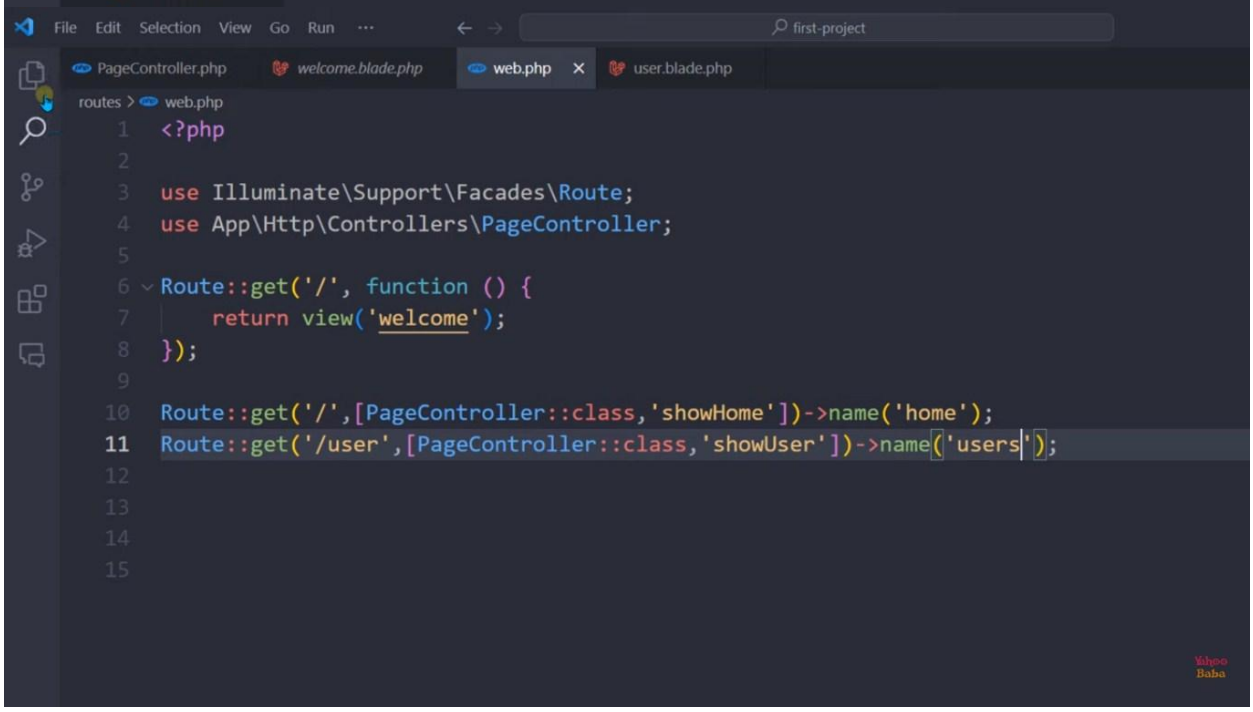
```
use App\Http\Controllers\UserController;

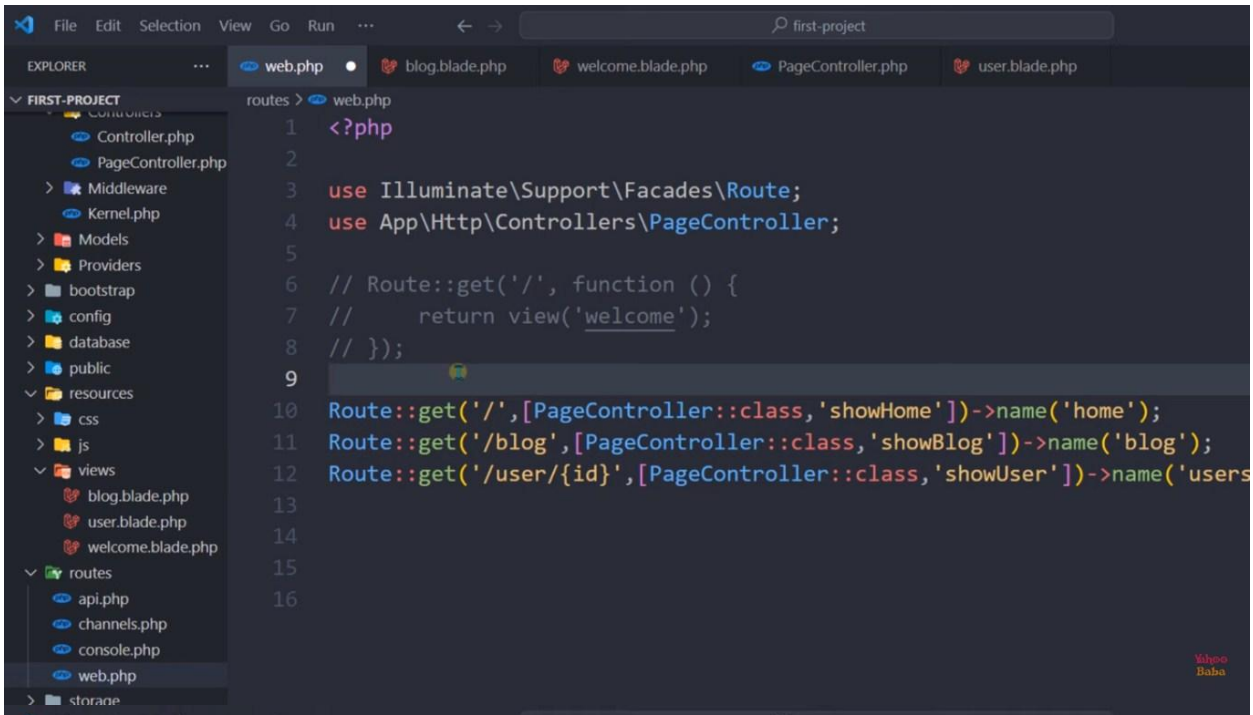
Route::get('/user', [UserController::class, 'show']);
```

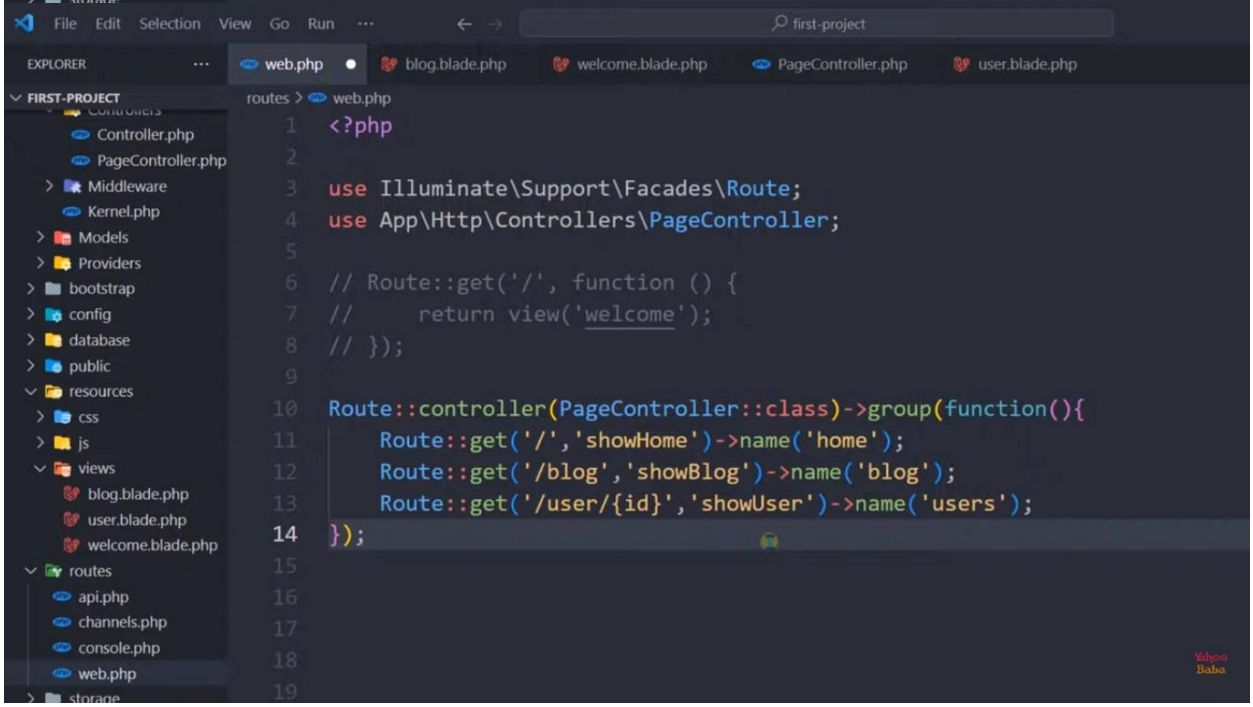
http://localhost/user




File Edit Selection View Go Run ···
first-project
EXPLORER
PageController.php
welcome.blade.php
web.php
FIRST-PROJECT
app
Console
Exceptions
Http
Controllers
Controller.php
PageController.php
Middleware
Kernel.php
Models
Providers
bootstrap
config
database
public
resources
css
js
views
welcome.blade.php
routes
api.php
channels.php
app > Http > Controllers > PageController.php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class PageController extends Controller
{
public function showHome(){
return view('welcome');
}
public function showUser(){
return view('user');
}
}


File Edit Selection View Go Run ···
first-project
PageController.php
welcome.blade.php
web.php
user.blade.php
routes > web.php
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\PageController;
Route::get('/', function () {
return view('welcome');
});
Route::get('/',[PageController::class,'showHome'])->name('home');
Route::get('/user',[PageController::class,'showUser'])->name('users');


File Edit Selection View Go Run
first-project
EXPLORER
web.php
blog.blade.php
welcome.blade.php
PageController.php
user.blade.php
FIRST-PROJECT
routes > web.php
Controller.php
PageController.php
Middleware
Kernel.php
Models
Providers
bootstrap
config
database
public
resources
css
js
views
blog.blade.php
user.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
1 <?php
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\PageController;
6 // Route::get('/', function () {
7 // return view('welcome');
8 // });
10 Route::get('/',[PageController::class,'showHome'])->name('home');
11 Route::get('/blog',[PageController::class,'showBlog'])->name('blog');
12 Route::get('/user/{id}',[PageController::class,'showUser'])->name('users


File Edit Selection View Go Run
first-project
EXPLORER
web.php
blog.blade.php
welcome.blade.php
PageController.php
user.blade.php
FIRST-PROJECT
routes > web.php
Controller.php
PageController.php
Middleware
Kernel.php
Models
Providers
bootstrap
config
database
public
resources
css
js
views
blog.blade.php
user.blade.php
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
1 <?php
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\PageController;
6 // Route::get('/', function () {
7 // return view('welcome');
8 // });
10 Route::controller(PageController::class)->group(function(){
11 Route::get('/','showHome')->name('home');
12 Route::get('/blog','showBlog')->name('blog');
13 Route::get('/user/{id}','showUser')->name('users');
14 });

Laravel : Single Action Controllers

```
class TestController extends Controller
{
    public function __invoke()
    {
        return view('test');
    }
}
```

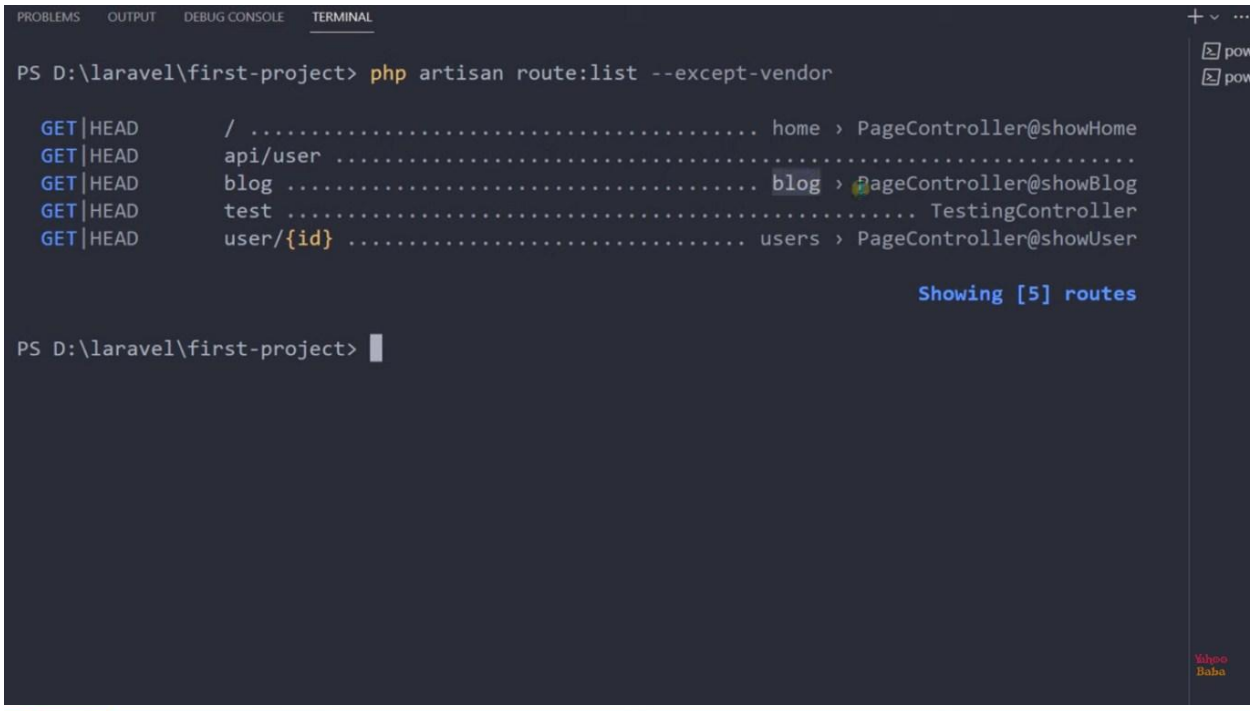
```
Route::get('/test', TestingController::class);
```


Laravel : Single Action Controllers

```
php artisan make:controller TestingController --invokable
or
php artisan make:controller TestingController --i
```

```
class TestController extends Controller
{
    public function __invoke()
    {
        return view('test');
    }
}
```

```
Route::get('/test', TestingController::class);
```



Laravel : Steps to work in Model

1. Create Database — CREATE DATABASE laravel-db or Xampp (Phpmyadmin)
   Projectfolder/.env
2. Create Database Migration (Create tables in database)
   php artisan make:migration create_students_table
   Projectfolder/database/migrations/
3. Seeding (Insert Initial Data in Tables)
4. Create Model

Laravel : Migration

```
CREATE TABLE table_name (
   id INT(),
   name VARCHAR(30),
   city VARCHAR(15),
  ....
);
```

1. `php artisan make:migration create_students_table`

2.
```
return new class extends Migration
{
    public function up(): void
    {
        Schema::create('students', function (Blueprint $table) {
            $table->id();
            $table->string('name', 30);
            $table->string('city', 15);
        });
    }
}
```

3. `php artisan migrate`

Laravel : Artisan Migration Commands

```
php artisan make:migration create_students_table
php artisan migrate

php artisan migrate:rollback
php artisan migrate:reset
php artisan migrate:refresh
```

Laravel : Add to Column with Migration

1. php artisan make:migration update_students_table --table=students

2.
```
return new class extends Migration
{
    public function up(): void
    {
        Schema::table('students', function (Blueprint $table) {
            $table->string('city');

        });
    }
}
```

3. php artisan migrate

Laravel : Modify Column with Migration

```
$table->renameColumn('from', 'to');
```

MySQL < 8.0.3
MariaDB < 10.5.2

```
$table->dropColumn('city');
```

```
$table->dropColumn(['city', 'avatar', 'location']);
```

```
$table->string('name', 50)->change();
```

```
$table->integer('votes')->unsigned()->default(1)->comment('my comment')->change();
```

Laravel : Constraints with Migration

MySQL

- NOT NULL
```
$table->string('email')->nullable();
```
- UNIQUE
```
$table->string('email')->unique();
$table->unique('email');
```
- DEFAULT
```
$table->string('city')->default('Agra');
```
- PRIMARY KEY
```
$table->primary('user_id');
```
- FOREIGN KEY
```
$table->foreign('user_id')->references('id')->on('users');
```
- CHECK

Laravel : Column Modifiers

Modifier	Description
->after('column')	Place the column "after" another column (MySQL).
->autoIncrement()	Set INTEGER columns as auto-incrementing (primary key).
->comment('my comment')	Add a comment to a column (MySQL/PostgreSQL).
->first()	Place the column "first" in the table (MySQL).
->from($integer)	Set the starting value of an auto-incrementing field (MySQL / PostgreSQL).
->invisible()	Make the column "invisible" to SELECT * queries (MySQL).
->unsigned()	Set INTEGER columns as UNSIGNED (MySQL).
->useCurrent()	Set TIMESTAMP columns to use CURRENT_TIMESTAMP as default value.
->useCurrentOnUpdate()	Set TIMESTAMP columns to use CURRENT_TIMESTAMP when a record is updated (MySQL).

Laravel : Primary Key & Foreign Key Constraints

Student Table

Id	Name	Age	City
1	Ram Kumar	19	1
2	Salman Khan	18	2
3	Meera Khan	19	1
4	Sarita Kumari	21	3

City Table

Cid	City
1	Agra
2	Bhopal
3	Delhi

Link

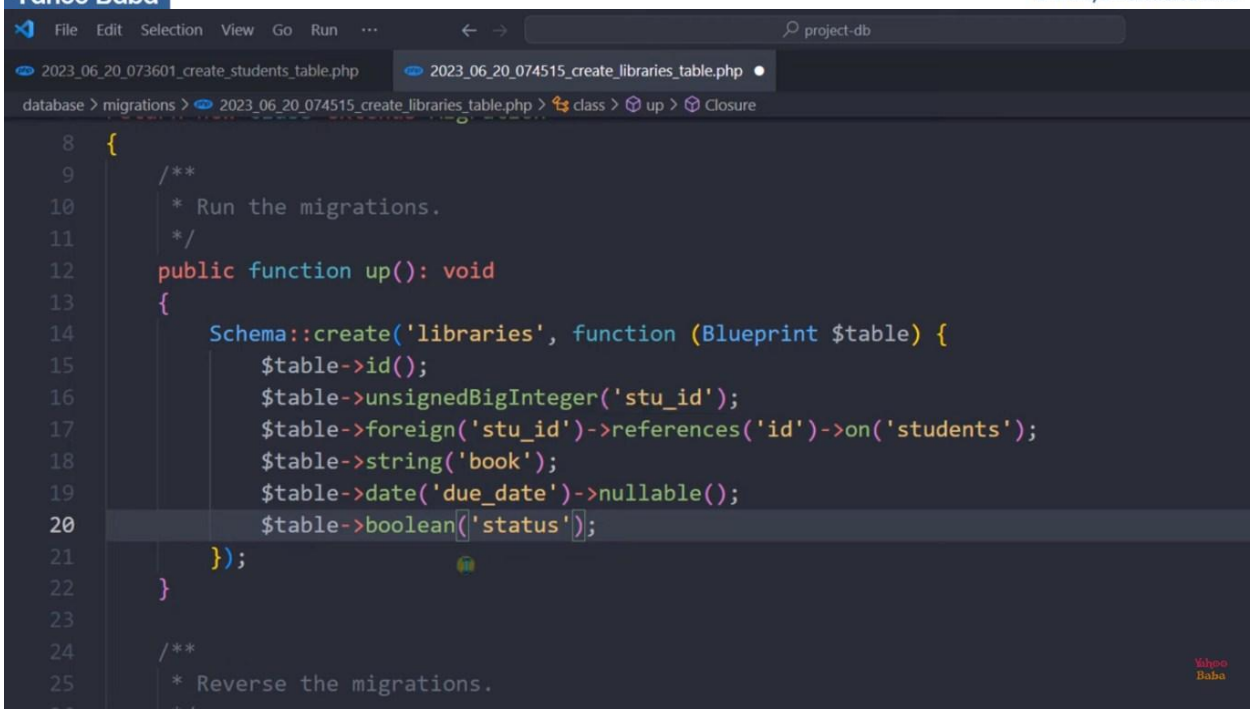
$table->primary('Cid');

PRIMARY KEY

FOREIGN KEY

PRIMARY KEY

$table->foreign('city')->references('Cid')->on('City');



Laravel : Steps to work in Seeder

1. php artisan make:model student
2. php artisan make:seeder StudentSeeder
3.

```
use App\Models\student;

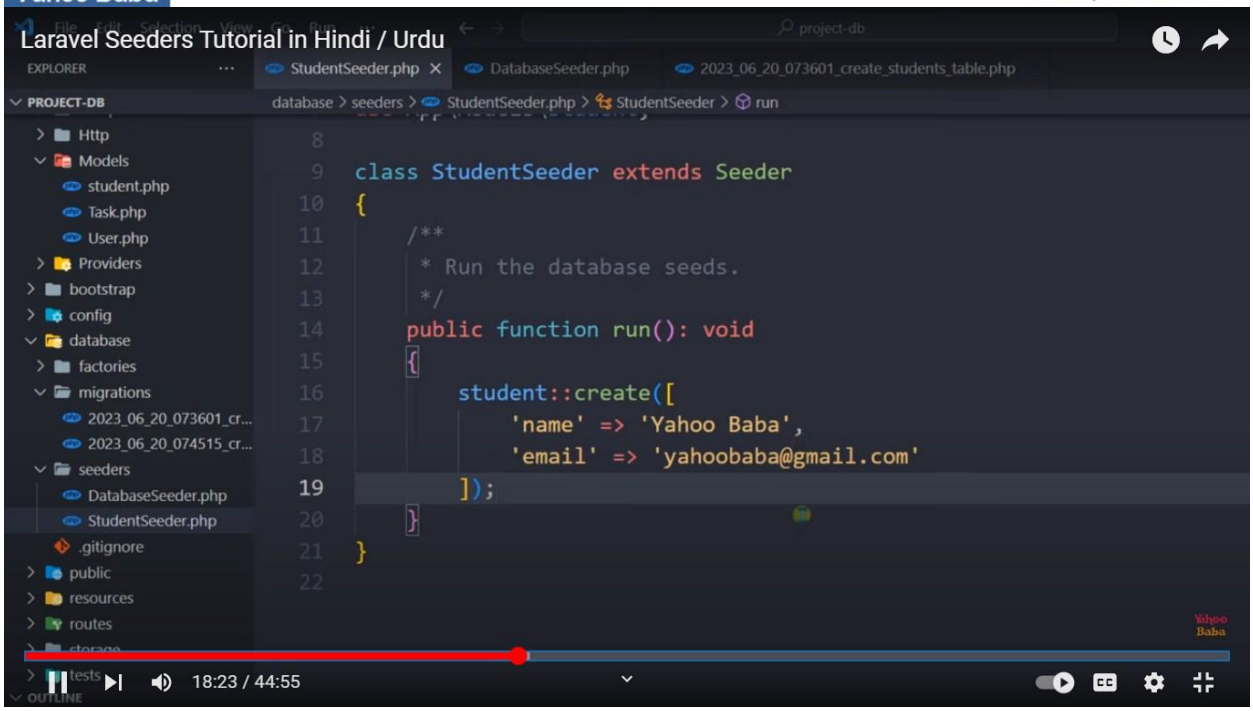
class StudentSeeder extends Seeder
{
    public function run(): void
    {
        student::create([
            'name' => 'Yahoo Baba',
            'email' => 'yahoobaba@gmail.com'
        ]);
    }
}
```

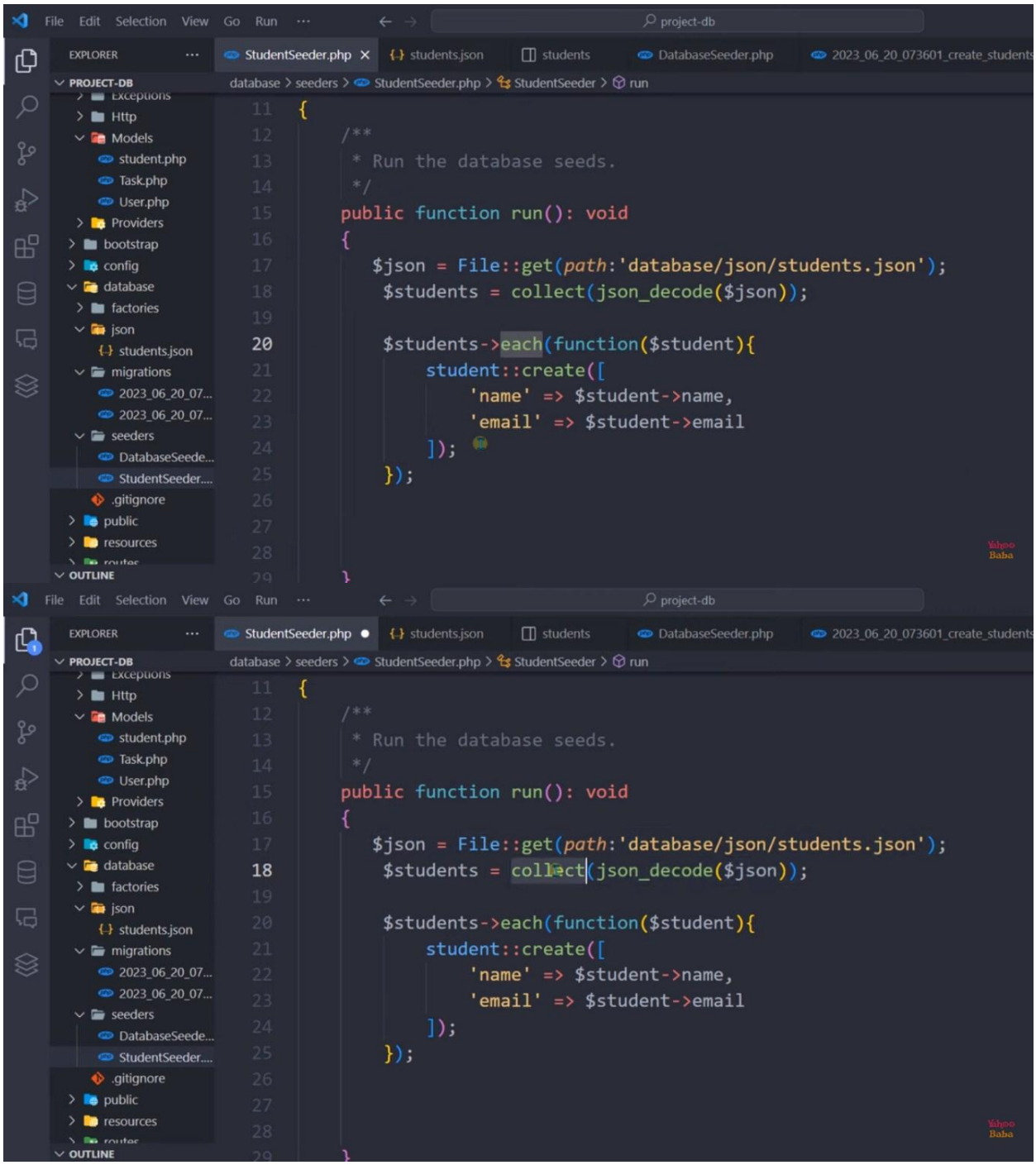
4. Seeders/DatabaseSeeder.php

```
$this->call([
    StudentSeeder::class
]);
```

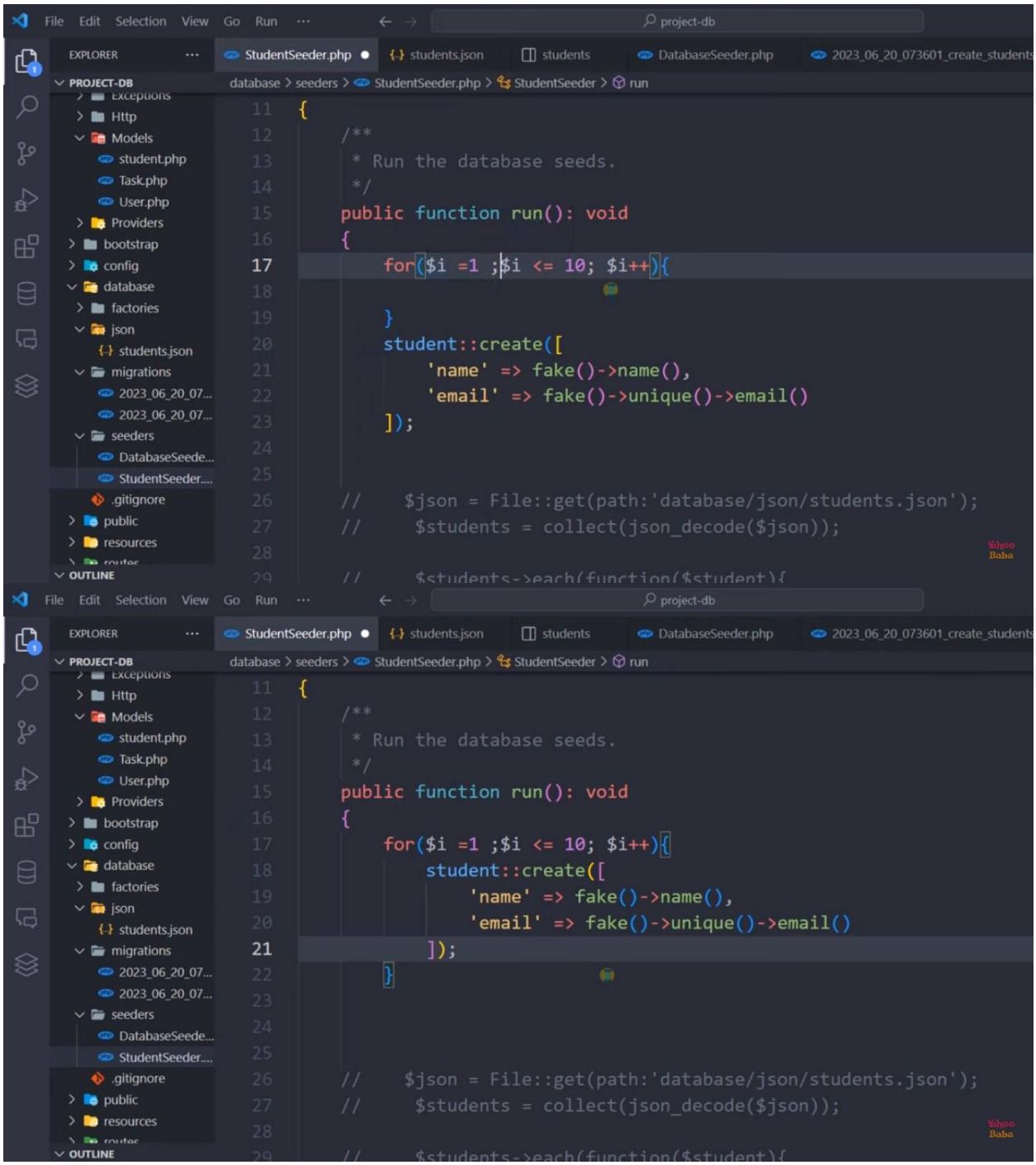
5. php artisan db:seed

Yahoo Baba www.yahoobaba.net





File Edit Selection View Go Run
project-db
EXPLORER
StudentSeeder.php
students.json
students
DatabaseSeeder.php
2023_06_20_073601_create_students
PROJECT-DB
database > seeders > StudentSeeder.php > StudentSeeder > run
Http
Models
student.php
Task.php
User.php
Providers
bootstrap
config
database
factories
json
students.json
migrations
seeders
DatabaseSeede...
StudentSeeder....
.gitignore
public
resources
OUTLINE
{
/**
* Run the database seeds.
*/
public function run(): void
{
$json = File::get(path:'database/json/students.json');
$students = collect(json_decode($json));
$students->each(function($student){
student::create([
'name' => $student->name,
'email' => $student->email
]);
});
}
Yahoo Baba


File Edit Selection View Go Run
project-db
EXPLORER
StudentSeeder.php
students.json
students
DatabaseSeeder.php
2023_06_20_073601_create_students
PROJECT-DB
database > seeders > StudentSeeder.php > StudentSeeder > run
Http
Models
student.php
Task.php
User.php
Providers
bootstrap
config
database
factories
json
students.json
migrations
seeders
DatabaseSeede...
StudentSeeder....
.gitignore
public
resources
OUTLINE
{
/**
* Run the database seeds.
*/
public function run(): void
{
for($i =1 ;$i <= 10; $i++){
}
student::create([
'name' => fake()->name(),
'email' => fake()->unique()->email()
]);
// $json = File::get(path:'database/json/students.json');
// $students = collect(json_decode($json));
Yahyaa Baba

File Edit Selection View Go Run
project-db
EXPLORER
StudentSeeder.php
students.json
students
DatabaseSeeder.php
2023_06_20_073601_create_students
PROJECT-DB
database > seeders > StudentSeeder.php > StudentSeeder > run
Http
Models
student.php
Task.php
User.php
Providers
bootstrap
config
database
factories
json
students.json
migrations
seeders
DatabaseSeede...
StudentSeeder....
.gitignore
public
resources
OUTLINE
{
/**
* Run the database seeds.
*/
public function run(): void
{
for($i =1 ;$i <= 10; $i++){
student::create([
'name' => fake()->name(),
'email' => fake()->unique()->email()
]);
}
// $json = File::get(path:'database/json/students.json');
// $students = collect(json_decode($json));
Yahyaa Baba

Laravel : Artisan Seeders Commands

```
php artisan make:seeder StudentSeeder
```

```
php artisan db:seed
```

```
php artisan db:seed --class=UserSeeder
```

```
php artisan db:seed --force
```

```
php artisan db:seed --class=UserSeeder --force
```


Laravel : Steps to work in Factory

1. `php artisan make:model student`
2. `php artisan make:factory StudentFactory`
3.
```
class StudentFactory extends Factory
{
    public function definition(): array
    {
        return[
            'name' => fake()->name(),
            'email' => fake()->email()
        ];
    }
}
```
4. Seeders/DatabaseSeeder.php
```
use App\Models\student;

student::factory()->count(5)->create();
```
5. `php artisan db:seed`

Laravel : Artisan Factory Commands

```
php artisan make:factory StudentFactory
```

```
php artisan make:factory StudentFactory --model=Student
```
(Factory & Model)

```
php artisan make:model student -f
```
(Model & Factory)

```
php artisan db:seed
```

```
php artisan db:seed --class=UserSeeder
```

```
php artisan migrate:fresh --seed
```
(Migration & Seeding)

Laravel : Steps to work in Query Builder

1. ```
   php artisan make:controller UserController
   ```

2. ```
   use Illuminate\Support\Facades\DB;

   class UserController extends Controller
   {
       public function show()
       {
           $users = DB::table('users')->get();
           return $users;
       }
   }
   ```

   - Read – get()
   - Insert – insert()
   - Update – update()
   - Delete – delete()

3. ```
   use App\Http\Controllers\UserController;

   Route::get('/user', [UserController::class, 'show']);
   ```

   http://localhost/user
   ```

Laravel : Read Data with Query Builder

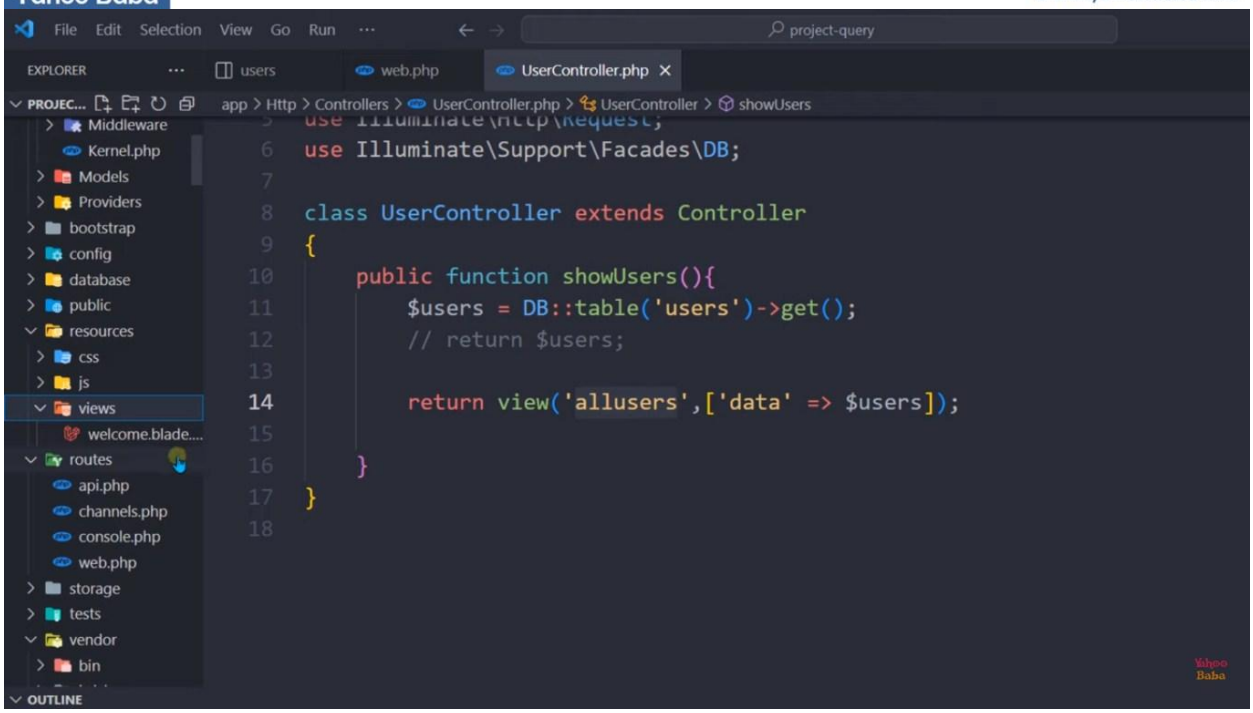
```
SELECT * FROM users
DB::table('users')->get()
```

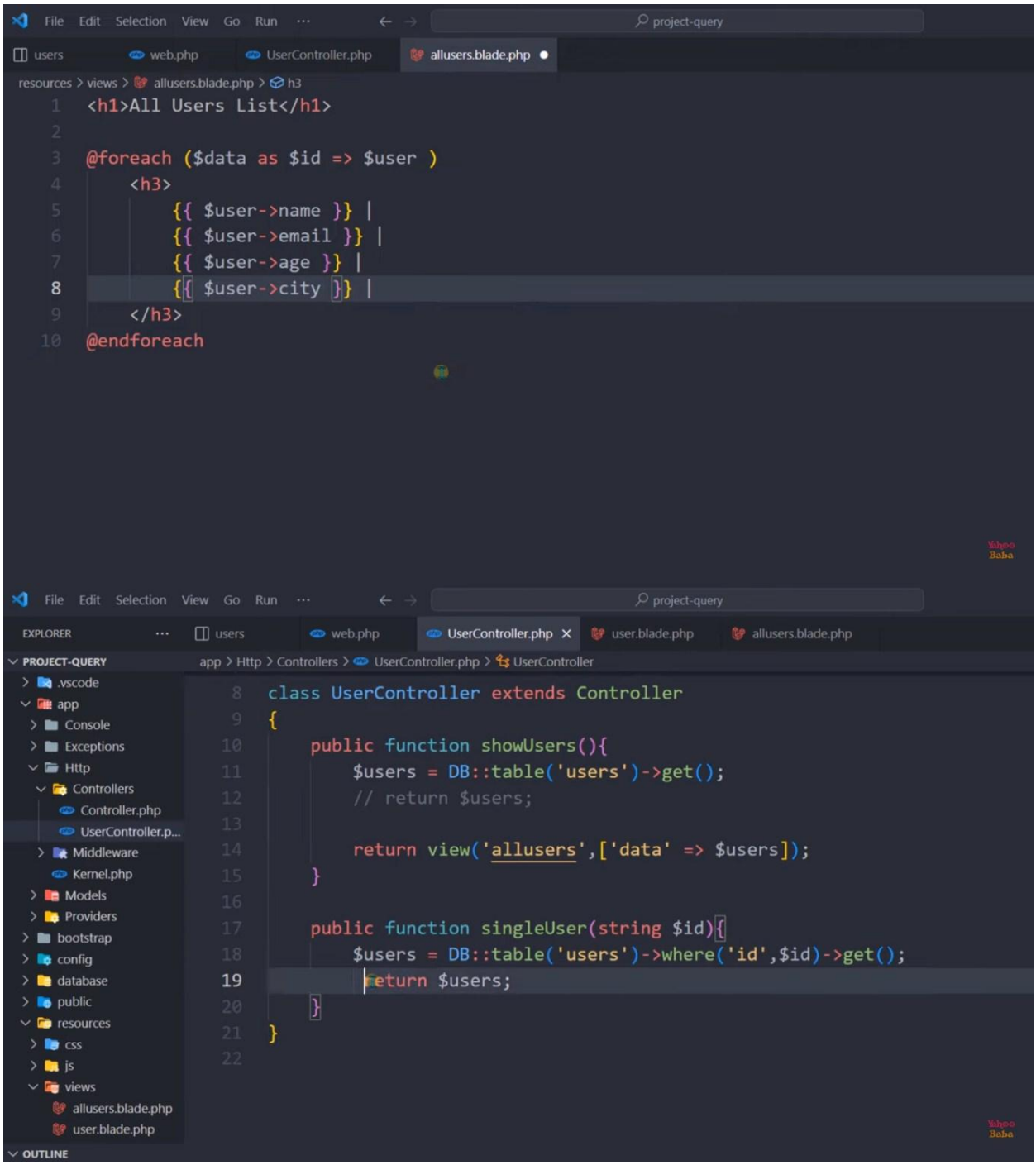
```
SELECT name, city FROM users
DB::table('users') ->select('name', 'city')->get()
```

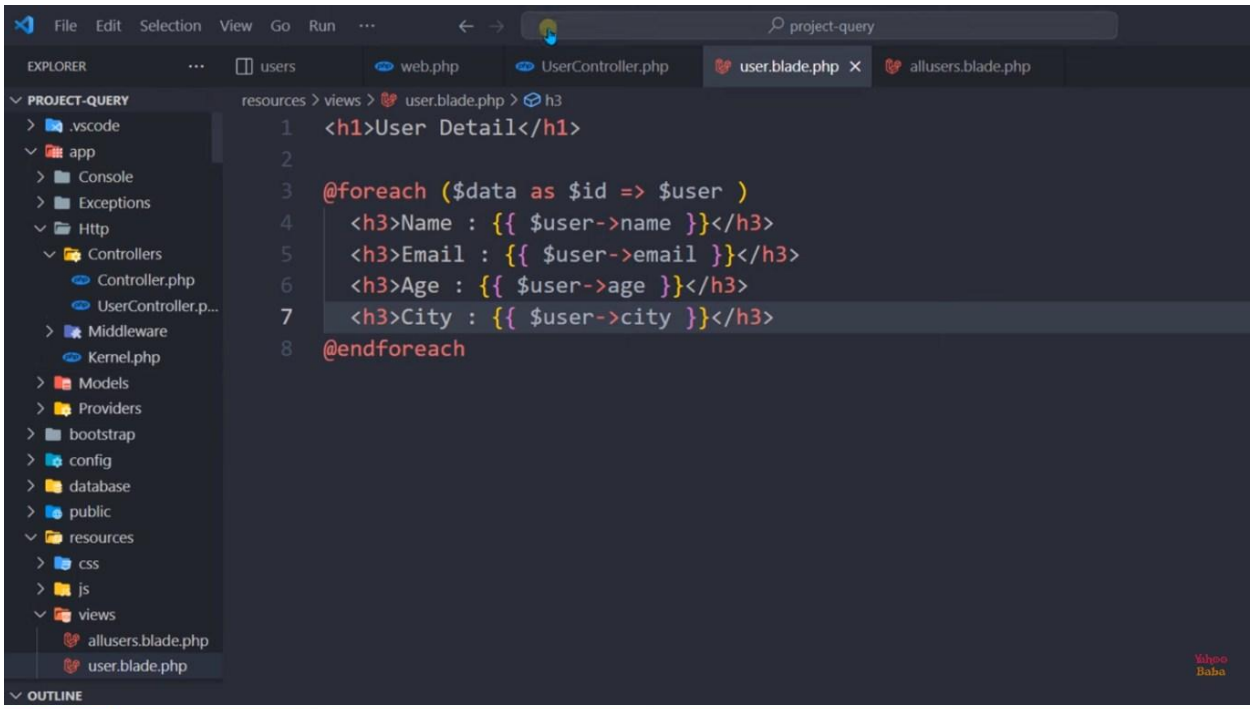
```
SELECT * FROM users WHERE city = 'goa';
DB::table('users') ->where('city', '=', 'goa')->get()
DB::table('users') ->where('city', '=', 'goa') ->where('age', '>', 18)->get()
DB::table('users') ->where('city', '=', 'goa') ->orWhere('age', '>', 18)->get()
```

- whereBetween()
- whereIn()
- whereNull()
- whereMonth()
- whereDay()
- whereYear()
- whereTime()

Yahoo Baba www.yahoobaba.net




File Edit Selection View Go Run
project-query
users
web.php
UserController.php
allusers.blade.php
resources > views > allusers.blade.php > h3
<h1>All Users List</h1>
@foreach ($data as $id => $user)
<h3>
{{ $user->name }} |
{{ $user->email }} |
{{ $user->age }} |
{{ $user->city }} |
</h3>
@endforeach
File Edit Selection View Go Run
project-query
EXPLORER
users
web.php
UserController.php
user.blade.php
allusers.blade.php
PROJECT-QUERY
app > Http > Controllers > UserController.php > UserController
.vscode
app
Console
Exceptions
Http
Controllers
Controller.php
UserController.p...
Middleware
Kernel.php
Models
Providers
bootstrap
config
database
public
resources
css
js
views
allusers.blade.php
user.blade.php
OUTLINE
class UserController extends Controller
{
public function showUsers(){
$users = DB::table('users')->get();
// return $users;
return view('allusers',['data' => $users]);
}
public function singleUser(string $id){
$users = DB::table('users')->where('id',$id)->get();
return $users;
}
}



Laravel : Query Builder

```
use Illuminate\Support\Facades\DB;
```

Add New Record

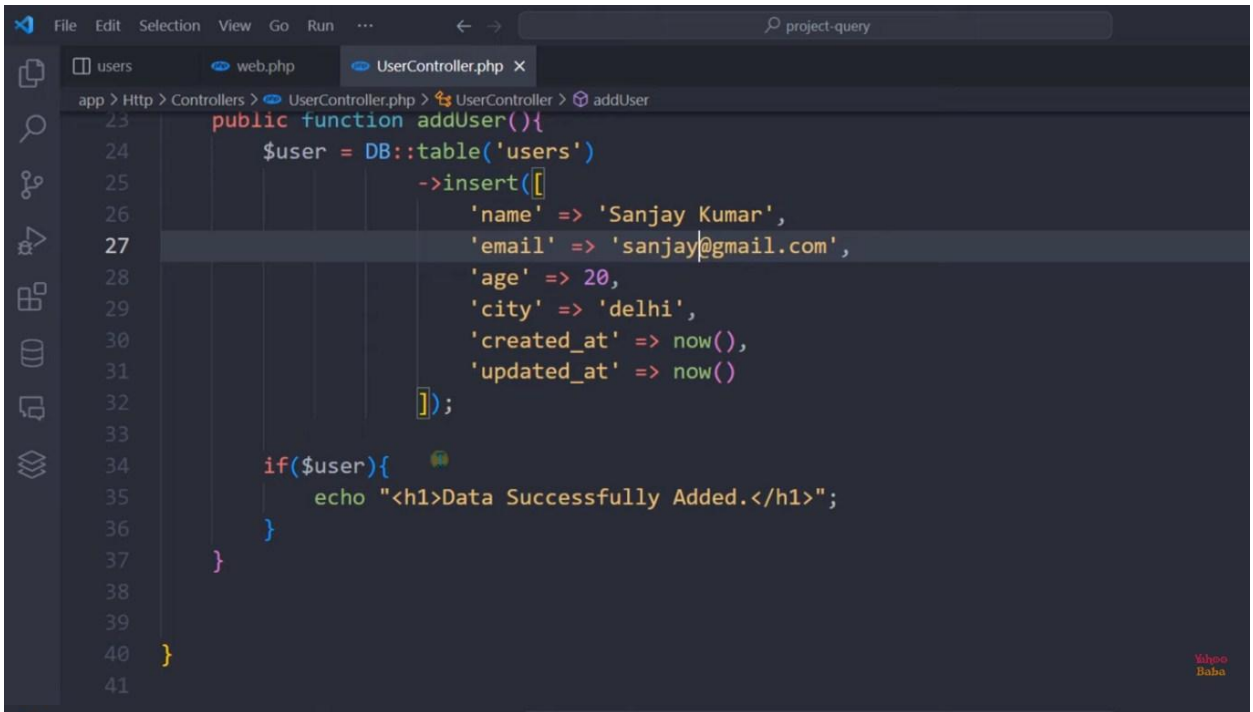
```
DB::table('users')->insert([
    'name' => 'Yahoo Baba',
    'email' => 'yahoobaba@email.com',
]);
```

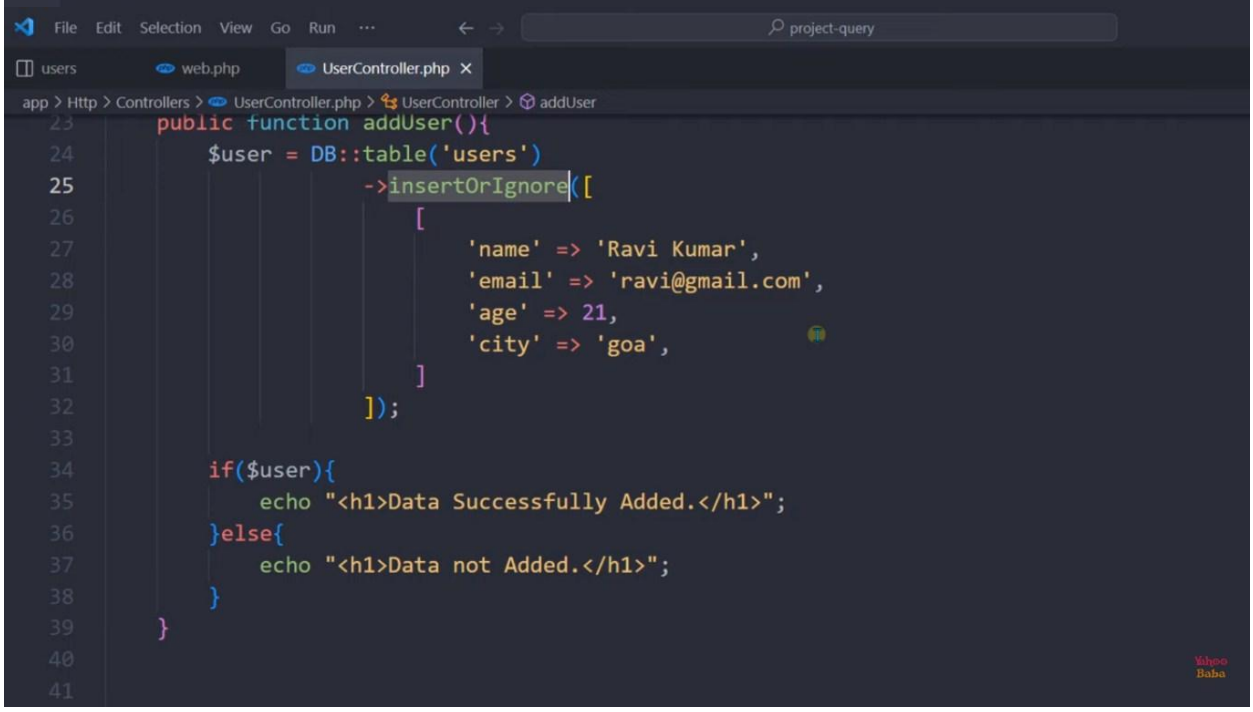
Delete Record

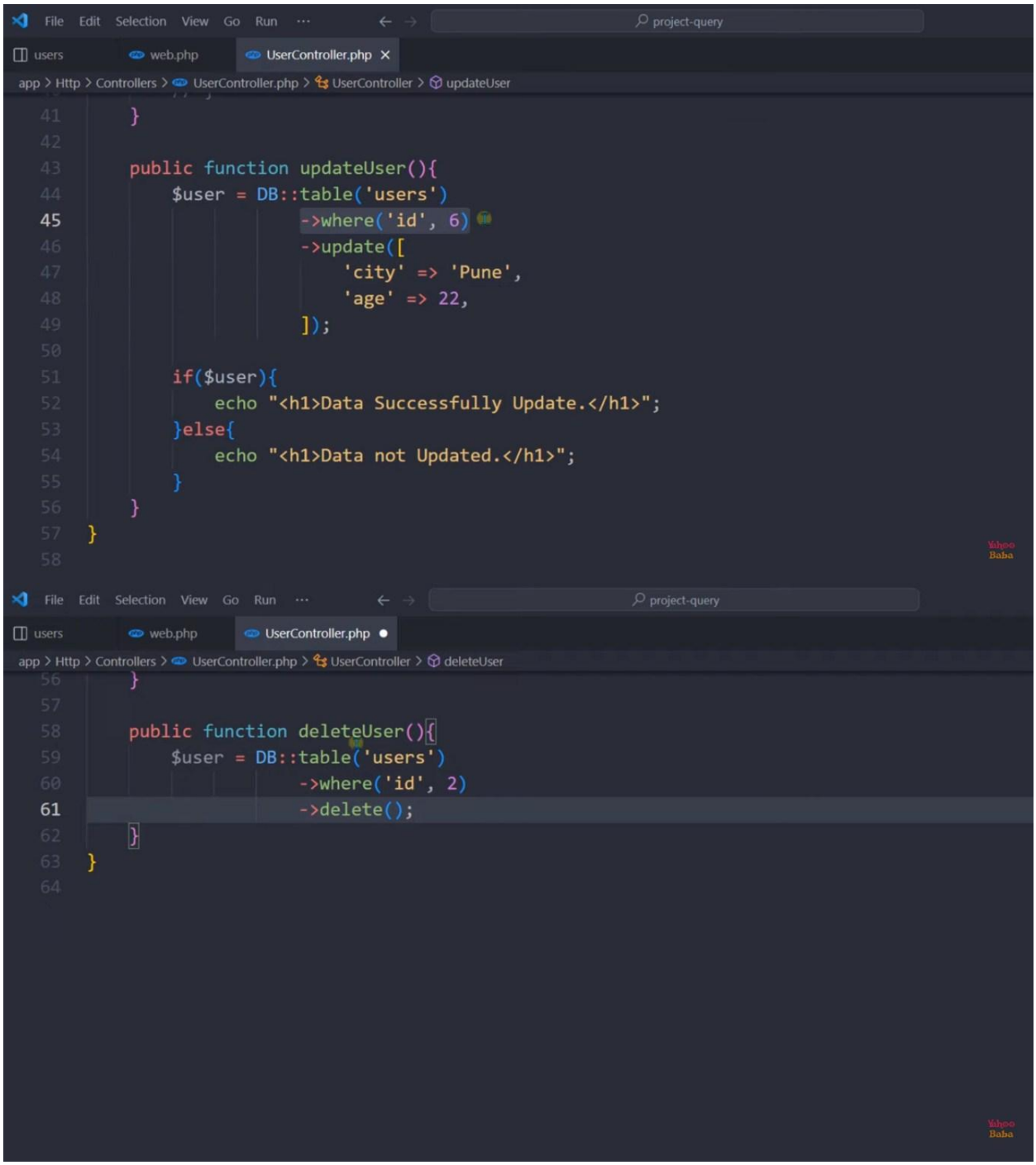
```
DB::table('users')
    ->where('id', 1)
    ->delete();
```

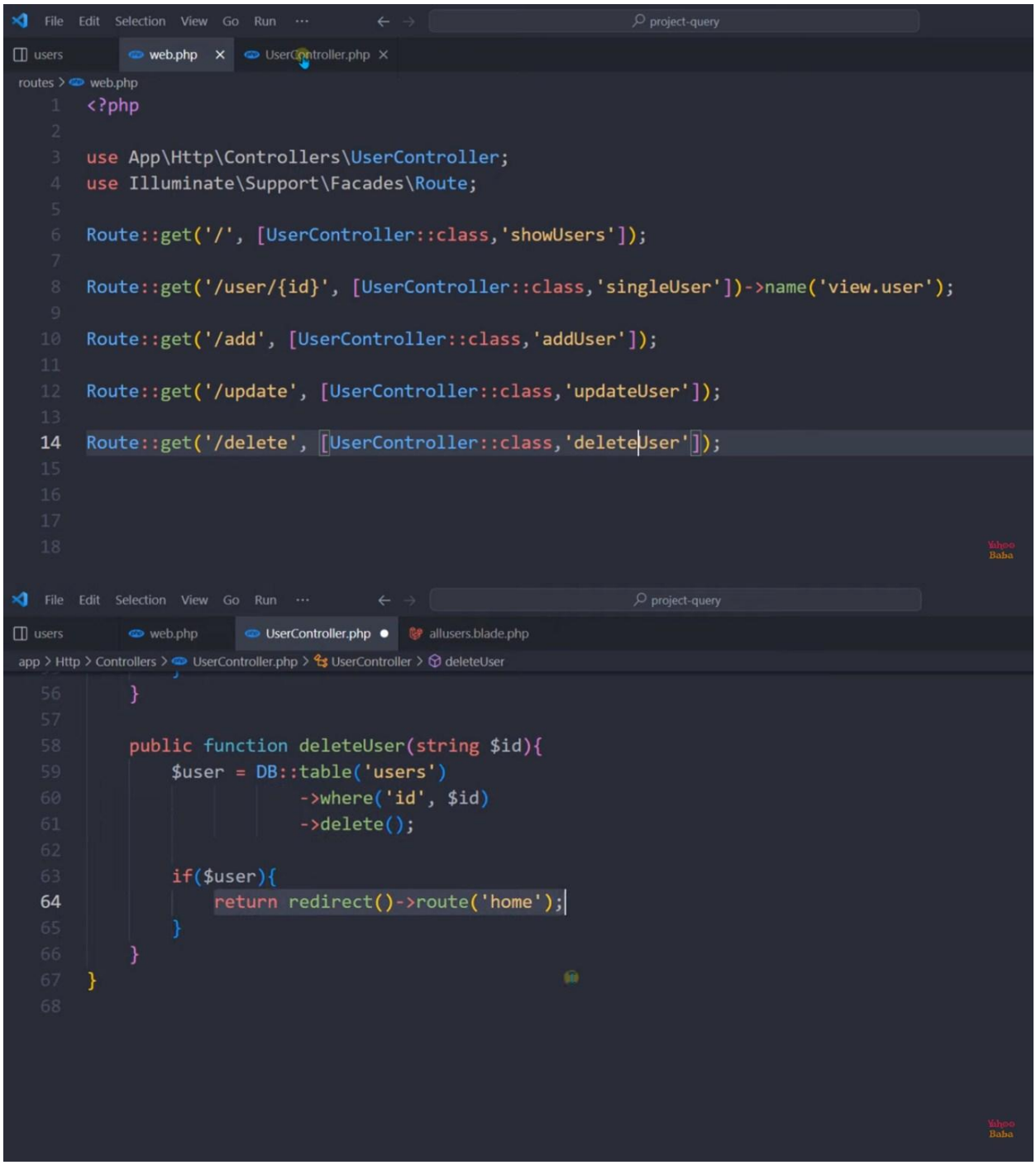
Update Existing Record

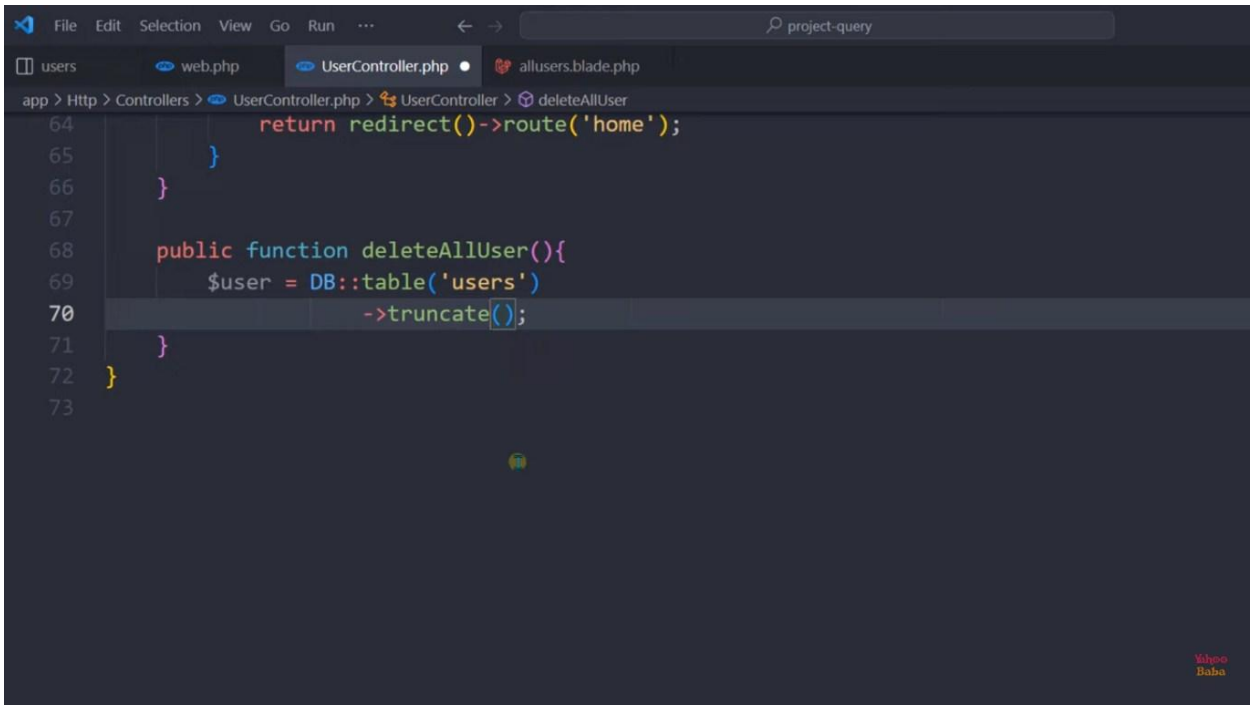
```
DB::table('users')
    ->where('id', 1)
    ->update([
        'city' => 'Agra',
    ]);
```


File Edit Selection View Go Run ···
project-query
users
web.php
UserController.php
app > Http > Controllers > UserController.php > UserController > addUser
public function addUser(){
$user = DB::table('users')
->insert([
'name' => 'Sanjay Kumar',
'email' => 'sanjay@gmail.com',
'age' => 20,
'city' => 'delhi',
'created_at' => now(),
'updated_at' => now()
]);
if($user){
echo "<h1>Data Successfully Added.</h1>";
}
}
}

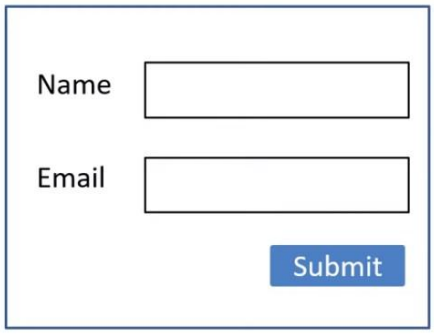

File Edit Selection View Go Run ···
project-query
users
web.php
UserController.php
app > Http > Controllers > UserController.php > UserController > addUser
public function addUser(){
$user = DB::table('users')
->insertOrIgnore([
[
'name' => 'Ravi Kumar',
'email' => 'ravi@gmail.com',
'age' => 21,
'city' => 'goa',
]
]);
if($user){
echo "<h1>Data Successfully Added.</h1>";
}else{
echo "<h1>Data not Added.</h1>";
}
}


File Edit Selection View Go Run ···
project-query
users
web.php
UserController.php
app > Http > Controllers > UserController.php > UserController > updateUser
}
public function updateUser(){
$user = DB::table('users')
->where('id', 6)
->update([
'city' => 'Pune',
'age' => 22,
]);
if($user){
echo "<h1>Data Successfully Update.</h1>";
}else{
echo "<h1>Data not Updated.</h1>";
}
}
}
File Edit Selection View Go Run ···
project-query
users
web.php
UserController.php
app > Http > Controllers > UserController.php > UserController > deleteUser
}
public function deleteUser(){
$user = DB::table('users')
->where('id', 2)
->delete();
}
}


File Edit Selection View Go Run
project-query
users
web.php
UserController.php
routes > web.php
<?php
use App\Http\Controllers\UserController;
use Illuminate\Support\Facades\Route;
Route::get('/', [UserController::class,'showUsers']);
Route::get('/user/{id}', [UserController::class,'singleUser'])->name('view.user');
Route::get('/add', [UserController::class,'addUser']);
Route::get('/update', [UserController::class,'updateUser']);
Route::get('/delete', [UserController::class,'deleteUser']);
Yahoo Baba
File Edit Selection View Go Run
project-query
users
web.php
UserController.php
allusers.blade.php
app > Http > Controllers > UserController.php > UserController > deleteUser
}
public function deleteUser(string $id){
$user = DB::table('users')
->where('id', $id)
->delete();
if($user){
return redirect()->route('home');
}
}
}
Yahoo Baba


File Edit Selection View Go Run ···
project-query
users
web.php
UserController.php
allusers.blade.php
app > Http > Controllers > UserController.php > UserController > deleteAllUser
return redirect()->route('home');
}
}
public function deleteAllUser(){
$user = DB::table('users')
->truncate();
}
}
Yahoo Baba

Laravel : Query Builder with HTML Forms




```
<form action="/addUser" method="POST">
    @csrf
    <input type="text" name="username">
    <input type="text" name="userpassword">
    <input type="submit">
</form>
```

```
Route::post('/add', 'addUser')
```

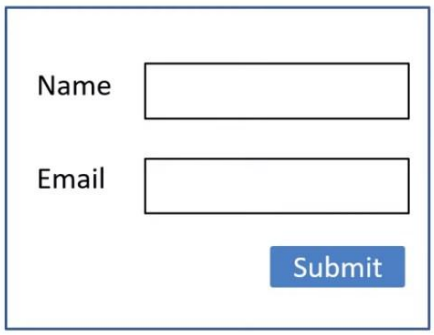
```
public function addUser(Request $req){

}
```

Cross-site Request Forgeries

malicious exploit

Laravel : Query Builder with HTML Forms




```
<form action="/addUser" method="POST">
    @csrf
    <input type="text" name="username">
    <input type="text" name="userpassword">
    <input type="submit">
</form>
```

```
Route::post('/add', 'addUser')
```

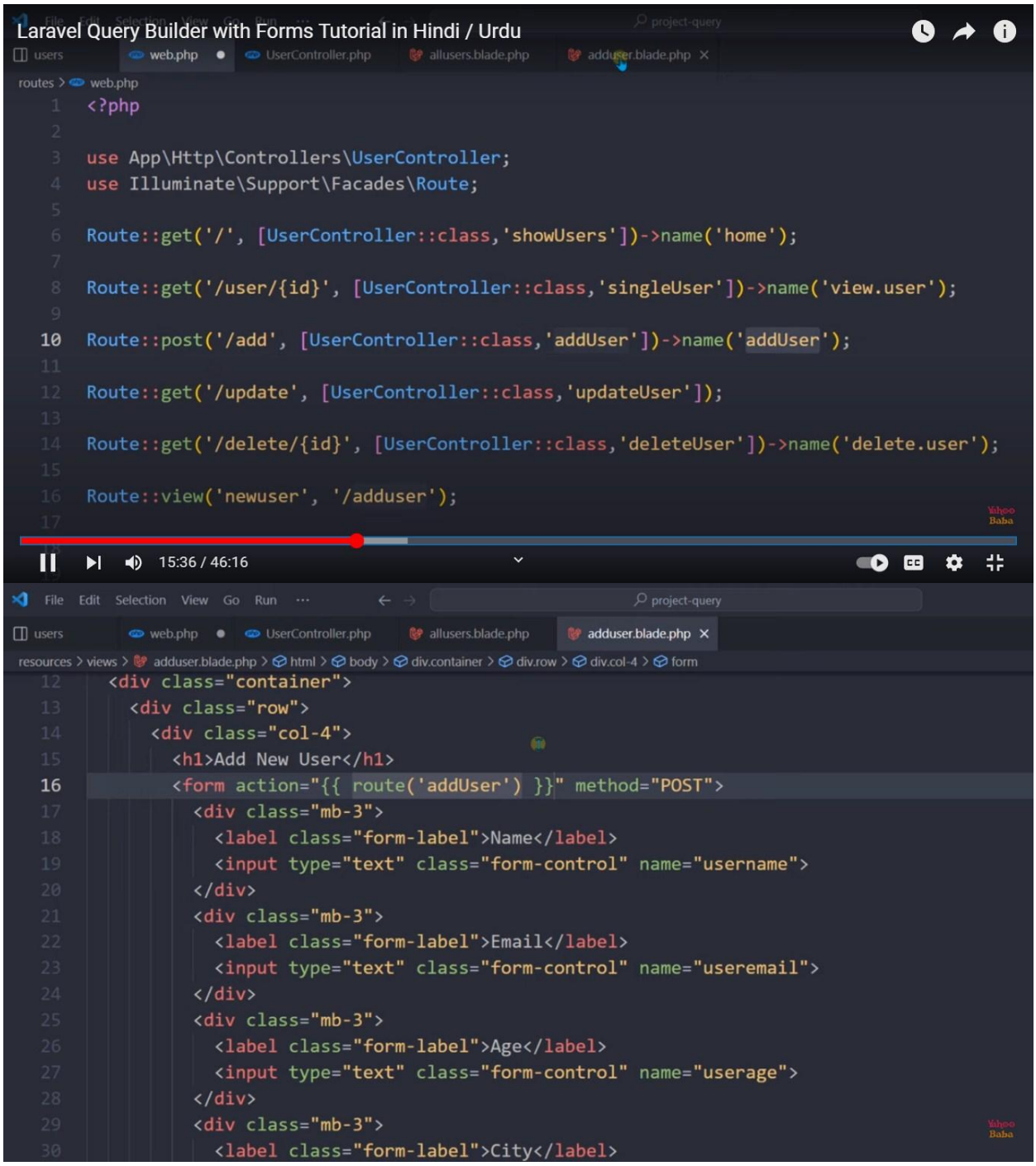
```
public function addUser(Request $req){

}
```

Cross-site Request Forgeries

malicious exploit

Yahoo Baba
www.yahoobaba.net


Laravel Query Builder with Forms Tutorial in Hindi / Urdu
users
web.php
UserController.php
allusers.blade.php
adduser.blade.php
routes > web.php
<?php
use App\Http\Controllers\UserController;
use Illuminate\Support\Facades\Route;
Route::get('/', [UserController::class,'showUsers'])->name('home');
Route::get('/user/{id}', [UserController::class,'singleUser'])->name('view.user');
Route::post('/add', [UserController::class,'addUser'])->name('addUser');
Route::get('/update', [UserController::class,'updateUser']);
Route::get('/delete/{id}', [UserController::class,'deleteUser'])->name('delete.user');
Route::view('newuser', '/adduser');
15:36 / 46:16
File Edit Selection View Go Run
project-query
resources > views > adduser.blade.php > html > body > div.container > div.row > div.col-4 > form

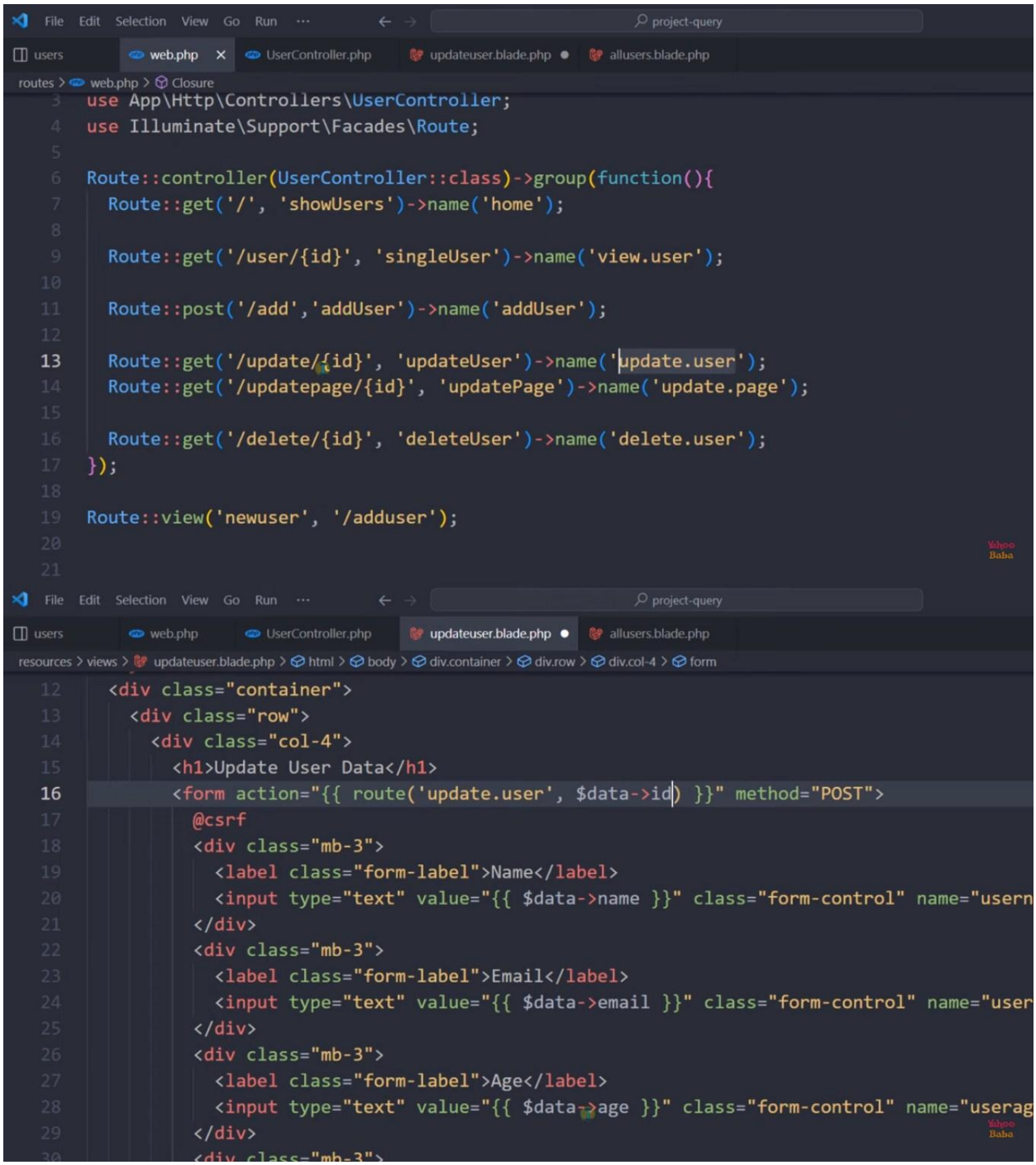
<h1>Add New User</h1>
<form action="{{ route('addUser') }}" method="POST">

<label class="form-label">Name</label>
<input type="text" class="form-control" name="username">

<label class="form-label">Email</label>
<input type="text" class="form-control" name="useremail">

<label class="form-label">Age</label>
<input type="text" class="form-control" name="userage">

<label class="form-label">City</label>

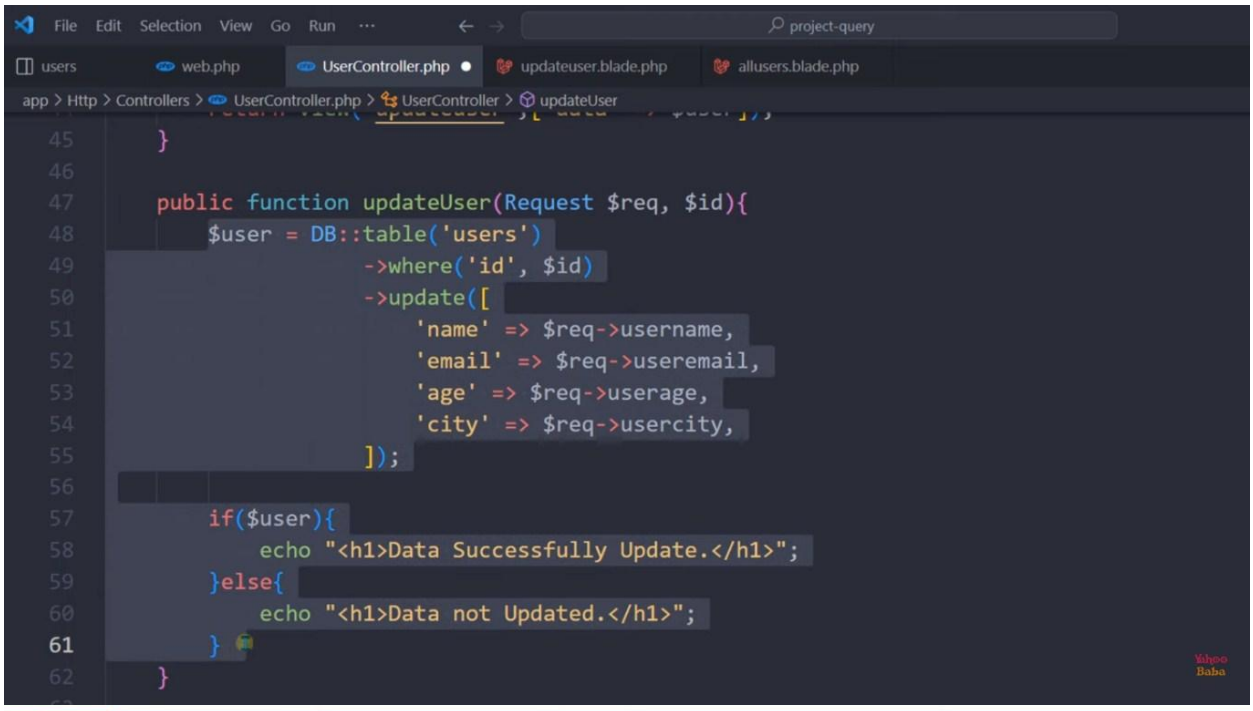

File Edit Selection View Go Run ···
project-query
users
web.php
UserController.php
updateuser.blade.php
allusers.blade.php
routes > web.php > Closure
use App\Http\Controllers\UserController;
use Illuminate\Support\Facades\Route;
Route::controller(UserController::class)->group(function(){
Route::get('/', 'showUsers')->name('home');
Route::get('/user/{id}', 'singleUser')->name('view.user');
Route::post('/add','addUser')->name('addUser');
Route::get('/update/{id}', 'updateUser')->name('update.user');
Route::get('/updatepage/{id}', 'updatePage')->name('update.page');
Route::get('/delete/{id}', 'deleteUser')->name('delete.user');
});
Route::view('newuser', '/adduser');
File Edit Selection View Go Run ···
project-query
users
web.php
UserController.php
updateuser.blade.php
allusers.blade.php
resources > views > updateuser.blade.php > html > body > div.container > div.row > div.col-4 > form

<h1>Update User Data</h1>
<form action="{{ route('update.user', $data->id) }}" method="POST">
@csrf

<label class="form-label">Name</label>
<input type="text" value="{{ $data->name }}" class="form-control" name="usern

<label class="form-label">Email</label>
<input type="text" value="{{ $data->email }}" class="form-control" name="user

<label class="form-label">Age</label>
<input type="text" value="{{ $data->age }}" class="form-control" name="userag



Laravel : Router Methods

Route::get(); → Read

Route::post(); → Add, Update, Delete

Route::put(); → Update

Route::patch(); → Update

Route::delete(); → Delete

Route::options();

```
<form method="POST">
    @csrf
    @method('PUT')

</form>
```

```
Route::match(['get', 'post'], '/', function () {
  // ...
});
```

```
Route::any('/', function () {
  // ...
});
```

Laravel : Pagination Methods

1. Paginate()

```
DB::table('users')->paginate(5)
```

< 1 2 3 4 >

2. simplePaginate()

```
DB::table('users')->simplePaginate(5)
```

Previous Next

Blade File :

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

3. cursorPaginate()

```
DB::table('users')->orderBy('id')->cursorPaginate(5)
```


Laravel : Form Validation

Name

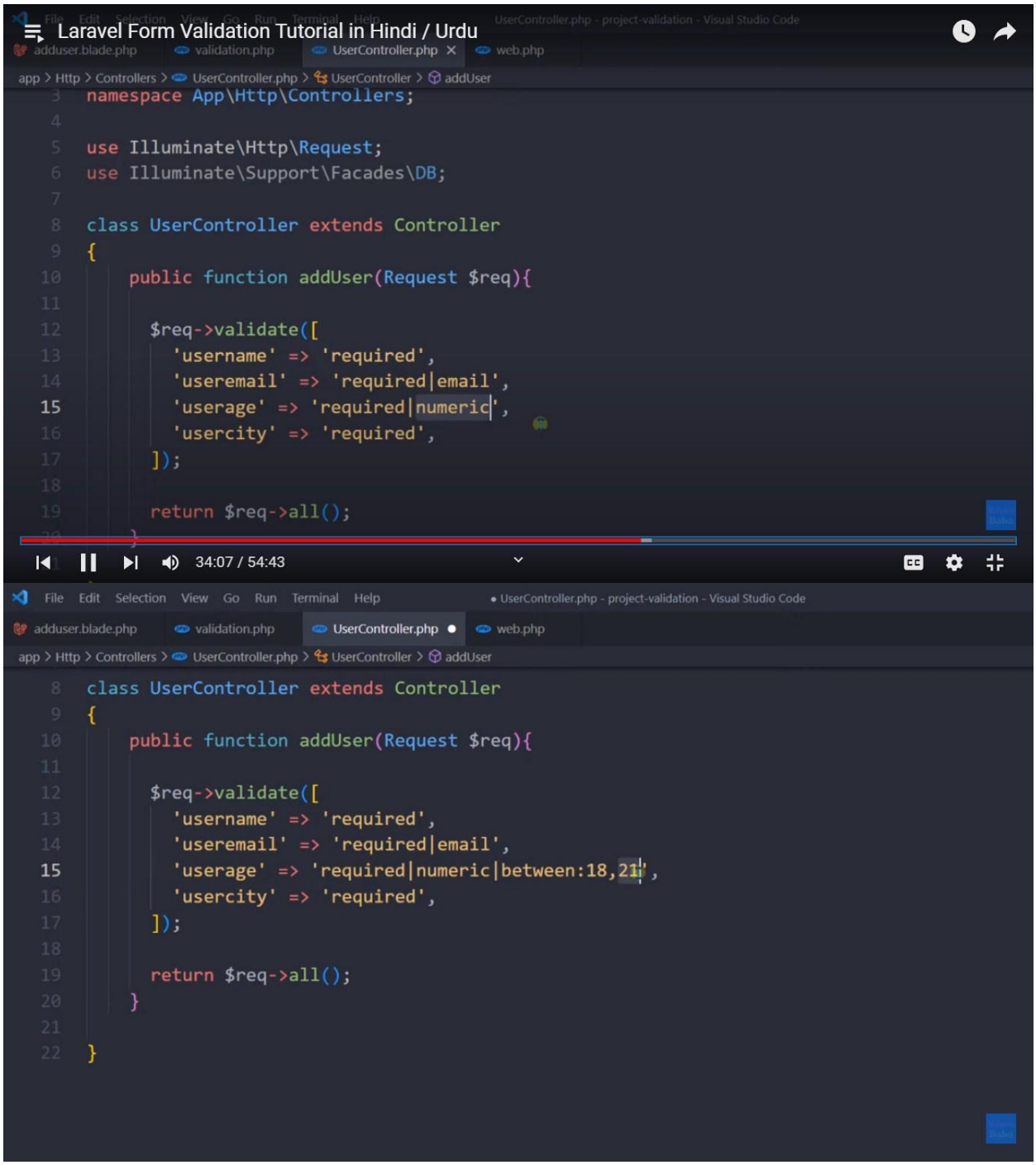
Email

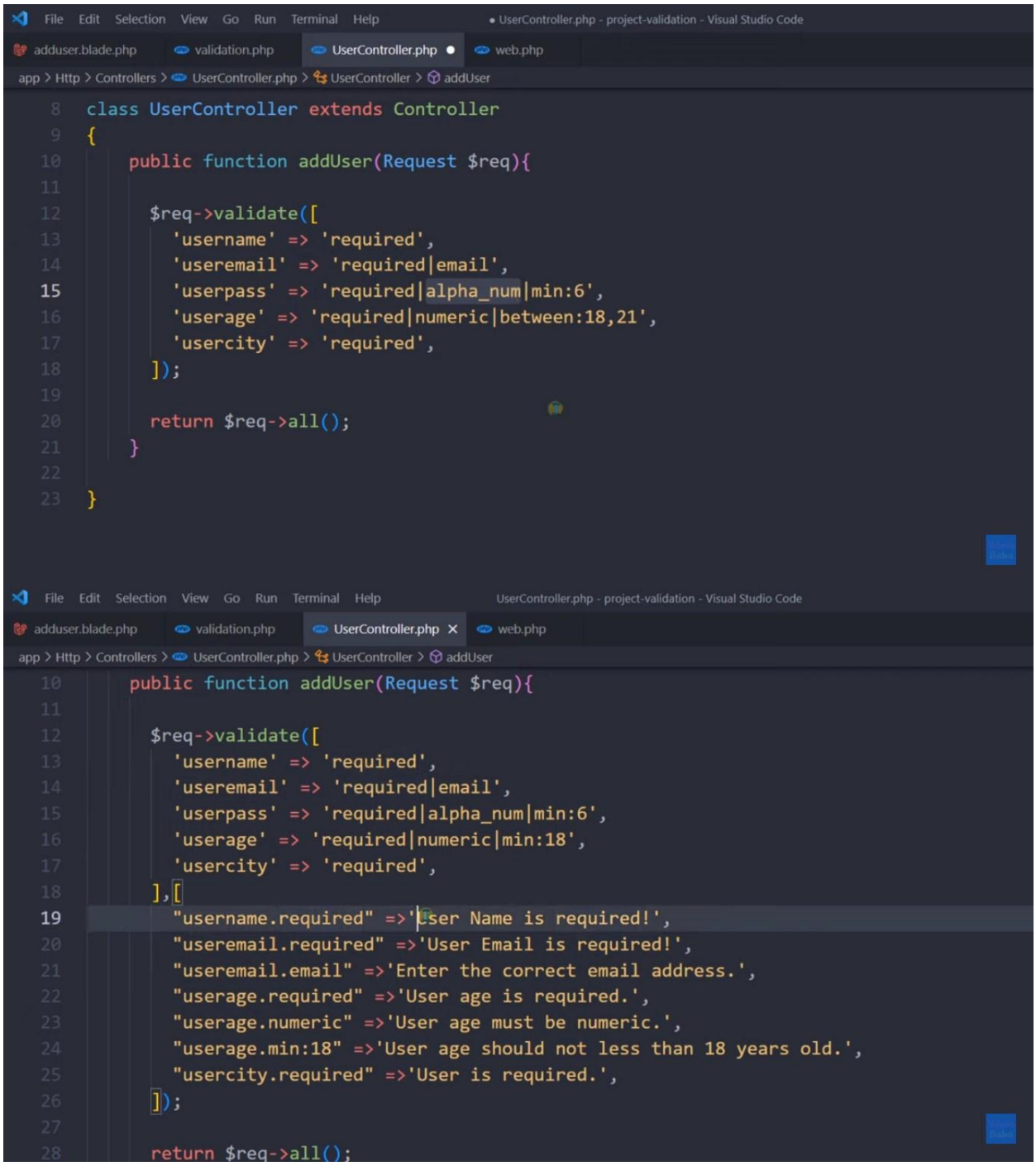
Submit

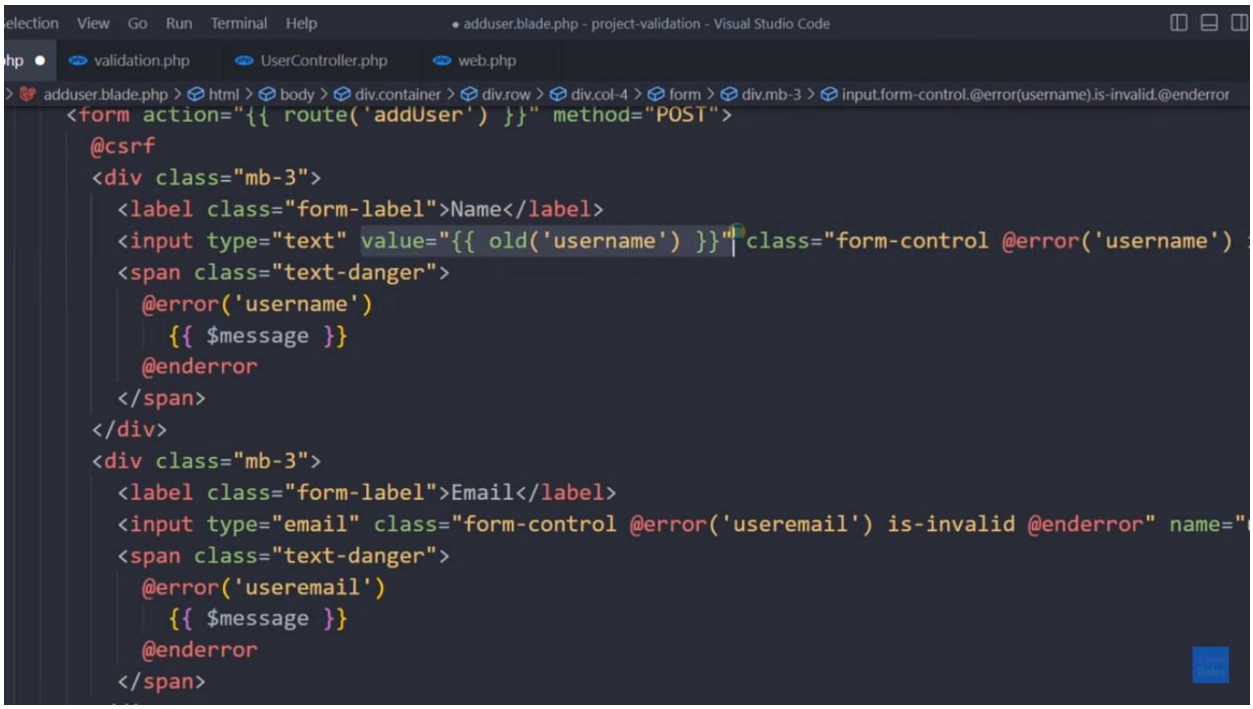
```
public function addUser(Request $req){

    $req->validate([
        'username' => 'required',
        'useremail' => 'required|email',
    ]);

}
```


Laravel Form Validation Tutorial in Hindi / Urdu
UserController.php - project-validation - Visual Studio Code
adduser.blade.php
validation.php
UserController.php
web.php
app > Http > Controllers > UserController.php > UserController > addUser
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
class UserController extends Controller
{
public function addUser(Request $req){
$req->validate([
'username' => 'required',
'useremail' => 'required|email',
'userage' => 'required|numeric',
'usercity' => 'required',
]);
return $req->all();
34:07 / 54:43
File Edit Selection View Go Run Terminal Help
• UserController.php - project-validation - Visual Studio Code
'userage' => 'required|numeric|between:18,21',


File Edit Selection View Go Run Terminal Help
• UserController.php - project-validation - Visual Studio Code
adduser.blade.php
validation.php
UserController.php
web.php
app > Http > Controllers > UserController.php > UserController > addUser
class UserController extends Controller
{
public function addUser(Request $req){
$req->validate([
'username' => 'required',
'useremail' => 'required|email',
'userpass' => 'required|alpha_num|min:6',
'userage' => 'required|numeric|between:18,21',
'usercity' => 'required',
]);
return $req->all();
}
}
File Edit Selection View Go Run Terminal Help
UserController.php - project-validation - Visual Studio Code
adduser.blade.php
validation.php
UserController.php
web.php
app > Http > Controllers > UserController.php > UserController > addUser
public function addUser(Request $req){
$req->validate([
'username' => 'required',
'useremail' => 'required|email',
'userpass' => 'required|alpha_num|min:6',
'userage' => 'required|numeric|min:18',
'usercity' => 'required',
],[
"username.required" =>'User Name is required!',
"useremail.required" =>'User Email is required!',
"useremail.email" =>'Enter the correct email address.',
"userage.required" =>'User age is required.',
"userage.numeric" =>'User age must be numeric.',
"userage.min:18" =>'User age should not less than 18 years old.',
"usercity.required" =>'User is required.',
]);
return $req->all();


Selection View Go Run Terminal Help
• adduser.blade.php - project-validation - Visual Studio Code
validation.php
UserController.php
web.php
adduser.blade.php > html > body > div.container > div.row > div.col-4 > form > div.mb-3 > input.form-control.@error(username).is-invalid.@enderror
<form action="{{ route('addUser') }}" method="POST">
@csrf

<label class="form-label">Name</label>
<input type="text" value="{{ old('username') }}" class="form-control @error('username')

@error('username')
{{ $message }}
@enderror

<label class="form-label">Email</label>
<input type="email" class="form-control @error('useremail') is-invalid @enderror" name="

@error('useremail')
{{ $message }}
@enderror

Laravel : Query Builder

```
use Illuminate\Support\Facades\DB;
```

Read Table Record

```
DB::table('users')->get();
```

Add New Record

```
DB::table('users')->insert([
    'name' => 'Yahoo Baba',
    'email' => 'yahoobaba@email.com',
]);
```

Update Existing Record

```
DB::table('users')
    ->where('id', 1)
    ->update([
        'city' => 'Agra',
    ]);
```

Delete Record

```
DB::table('users')
    ->where('id', 1)
    ->delete();
```


Laravel : Raw SQL Queries

```
use Illuminate\Support\Facades\DB;
```

```
DB::select('select * from users')
```

```
DB::select('select * from users where id = ?', [2])
```

```
DB::insert('insert into students (name, age) values (?, ?)',["Ram Kumar", 20])
```

```
DB::update('update users set email = "test@gmail.com" where id = ?',[2])
```

Laravel : Raw methods for Query Builder

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

```
DB::table('users')
    ->selectRaw('name, age')
    ->get();
```

```
DB::table('users')
   ->where ('age', '>', '20')
   ->get();
```

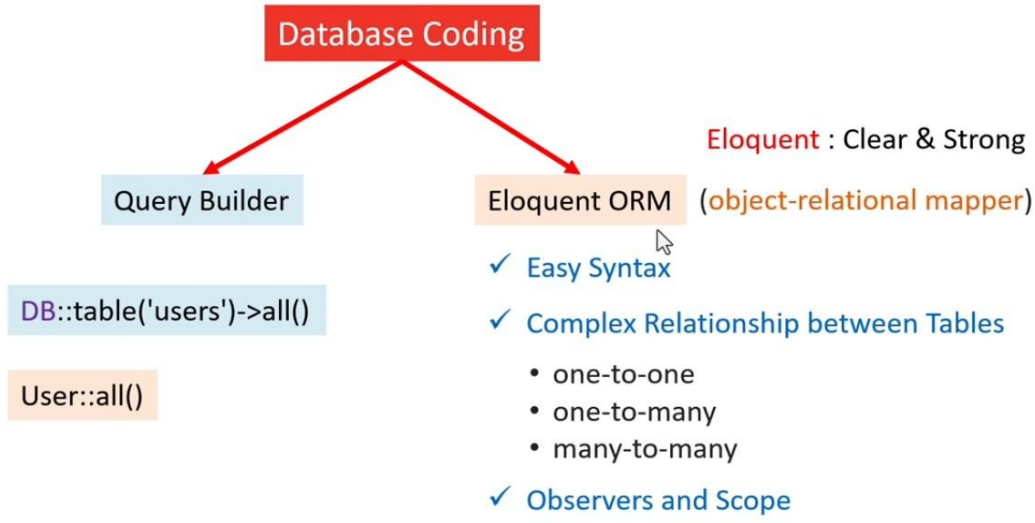
```
DB::table('users')
   ->whereRaw('age > 20')
   ->get();
```

```
->whereRaw('age > ?',[20])
```

```
DB::table('users')
   ->orderBy ('age', 'city')
   ->get();
```

```
DB::table('users')
   ->orderByRaw ('age, city')
   ->get();
```

Laravel : Database Coding



Laravel : Steps to work in Eloquent ORM

1. php artisan make:model user --controller

2.
```
use App\Models\User;

class UserController extends Controller
{
    public function show()
    {
        $users = User::all();
        return $users;
    }
}
```

- Read – get()
- Insert – create()
- Update – update()
- Delete – delete()

3.
```
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'show']);
```

http://localhost/user

Laravel : Read Data with Eloquent ORM

```
SELECT * FROM users
User::all()

SELECT name, city FROM users
User::select('name', 'city')->get()

SELECT * FROM users WHERE city = 'goa'
User::where('city', '=', 'goa')->get()
User::where('city', '=', 'goa') ->where('age', '>', 18)->get()
User::where('city', '=', 'goa') ->orWhere('age', '>', 18)->get()
```

- whereNot()
- whereBetween()
- whereIn()
- whereNull()
- whereNotNull()
- whereMonth()
- whereDate()
- whereDay()
- whereYear()
- whereTime()

Laravel : Raw methods for Eloquent ORM

```
user::select('name', 'age')
    ->get();

user::selectRaw('name, age')
    ->get();

user::where ('age', '>', '20')
    ->get();

user::whereRaw('age > 20')
    ->get();

whereRaw('age > ?',[20])

user::orderBy ('age', 'city')
    ->get();

user::orderByRaw ('age, city')
    ->get();
```

Laravel : Determining If Records Exist

```php
if (User::where('id', 1)->exists()) {
    // ...
}
```

```php
if (User::where('id', 1)->doesntExist()) {
    // ...
}
```


Laravel : Join Tables with Eloquent ORM

- INNER JOIN `join()`
- LEFT JOIN `leftJoin()`
- RIGHT JOIN `rightJoin()`
- CROSS JOIN `crossJoin()`

- Union()
- When()
- Chunk()

Laravel : Create Data with Eloquent ORM

Method : I

```
use App\Models\User;

$user = new User;

$user->name = 'Yahoo Baba';
$user->email = 'yahoobaba@email.com';

$user->save();
```

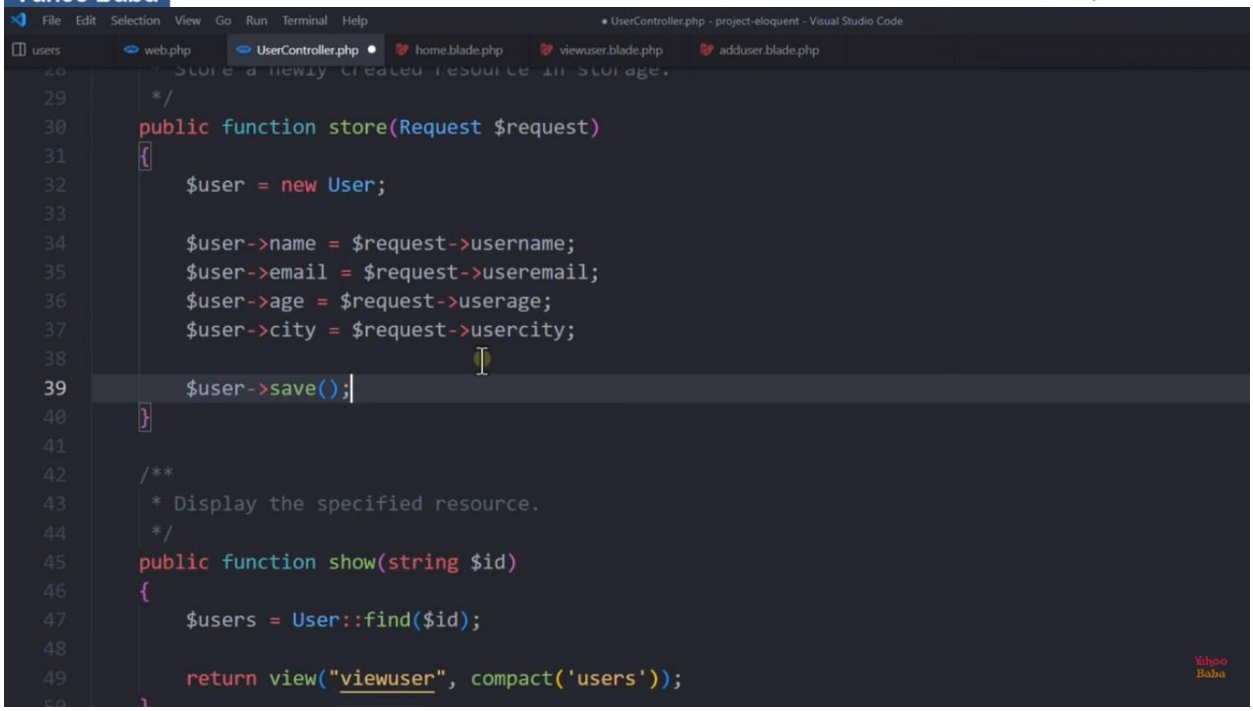
Method : II (Mass Assignment)

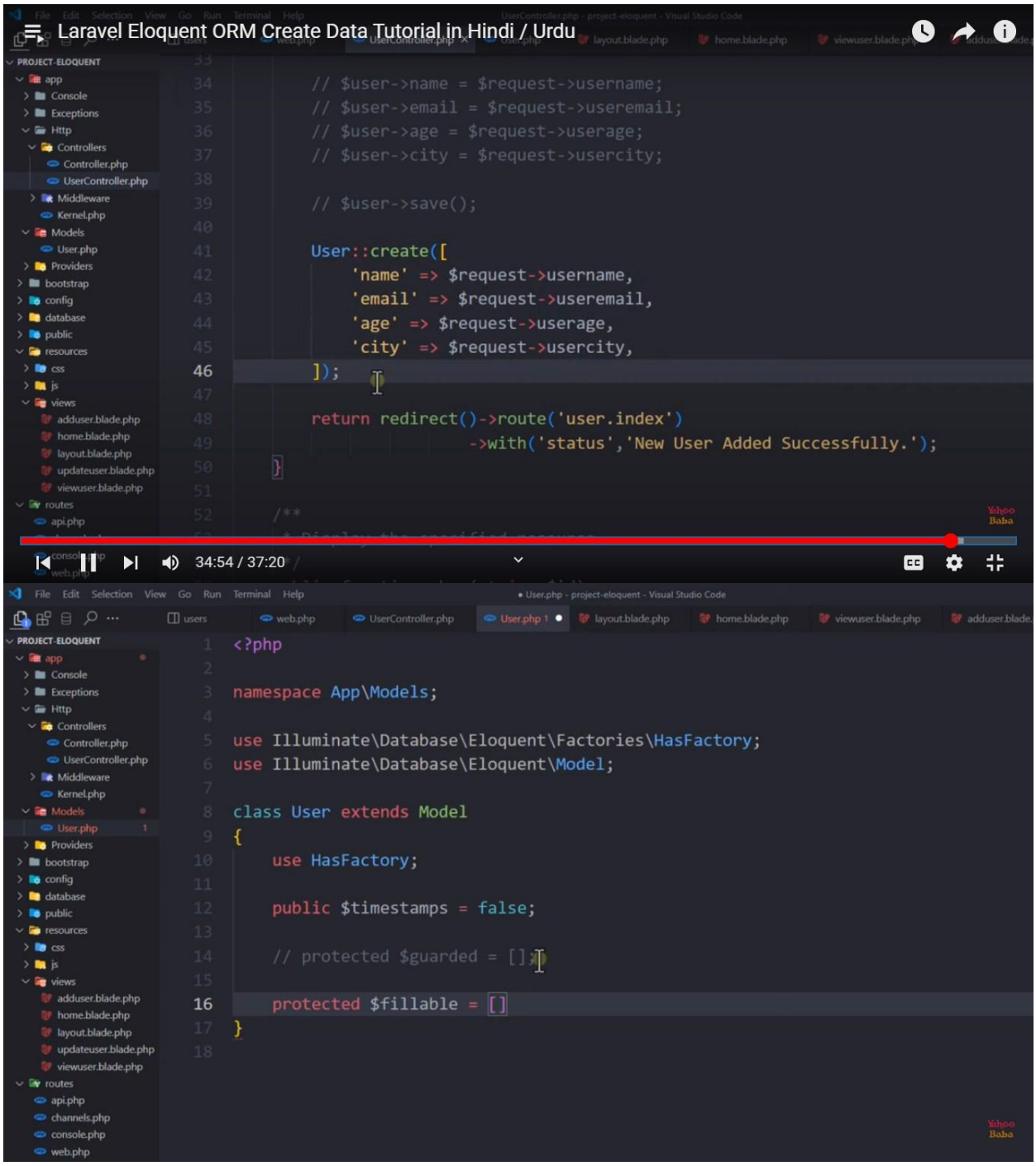
```
use App\Models\User;

User::create([
    'name' => 'Yahoo Baba',
    'email' => 'yahoobaba@email.com',
]);
```

Models/User.php

```
protected $guarded = [];

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

Yahoo Baba www.yahoobaba.net




Laravel Eloquent ORM Create Data Tutorial in Hindi / Urdu
PROJECT-ELOQUENT
app
Console
Exceptions
Http
Controllers
Controller.php
UserController.php
Middleware
Kernel.php
Models
User.php
Providers
bootstrap
config
database
public
resources
css
js
views
adduser.blade.php
home.blade.php
layout.blade.php
updateuser.blade.php
viewuser.blade.php
routes
api.php
// $user->name = $request->username;
// $user->email = $request->useremail;
// $user->age = $request->userage;
// $user->city = $request->usercity;
// $user->save();
User::create([
'name' => $request->username,
'email' => $request->useremail,
'age' => $request->userage,
'city' => $request->usercity,
]);
return redirect()->route('user.index')
->with('status','New User Added Successfully.');
}
/**
34:54 / 37:20
Yahoo Baba
File Edit Selection View Go Run Terminal Help
User.php - project-eloquent - Visual Studio Code
users
web.php
UserController.php
User.php 1
layout.blade.php
home.blade.php
viewuser.blade.php
adduser.blade.
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
class User extends Model
{
use HasFactory;
public $timestamps = false;
// protected $guarded = [];
protected $fillable = []
}
channels.php
console.php
Yahoo Baba

Laravel : Update Data with Eloquent ORM

Method : I

```
use App\Models\User;

$user = User::find(2);

$user->email = 'yahoobaba@email.com';

$user->save();
```

Method : II (Mass Update)

```
use App\Models\User;

User::find(2)->update([
    'email' => 'yahoobaba@email.com',
]);

User::where('name', 'Yahoo Baba')
    ->update([
    'email' => 'yahoobaba@email.com',
]);
```


Laravel : Delete Data with Eloquent ORM

Method : I

```
use App\Models\User;

$user = User::find(2);

$user->delete();
```

Method : III

```
User::truncate();
```

- Delete all data in database table.
- Reset the auto-incrementing ID

Method : II (Mass Delete with IDs)

```
use App\Models\User;

User::destroy(1);

User::destroy(1, 2, 3);

User::destroy([1, 2, 3]);
```

Laravel : Type of Controllers

1. Basic Controller

```
php artisan make:controller UserController
```

2. Single Action Controller

```
php artisan make:controller UserController --invokable
```

3. Resource Controller (Create, Read, Update, Delete)

```
php artisan make:controller UserController --resource
```


Laravel : Resource Controller

UserController.php

```
class UserController extends Controller
{
    public function index(){ }

    public function create(){ }

    public function store(Request $request){ }

    public function show(User $user){ }

    public function edit(User $User){ }

    public function update(Request $request, User $user){ }

    public function destroy(User $user){ }
}
```

Laravel : Route for Resource Controller

routes/web.php

```
use App\Http\Controllers\UserController;

Route::resource('users', UserController::class);
```

Verb	URI	Action	Route Name
GET	/users	index	users.index
GET	/users/create	create	users.create
POST	/users	store	users.store
GET	/users/{user}	show	users.show
GET	/users/{user}/edit	edit	users.edit
PUT/PATCH	/users/{user}	update	users.update
DELETE	/users/{user}	destroy	users.destroy


```
21  Route::resource('users',UserController::class);
```

```
17  // Route::get('/', function () {
18  //     return view('welcome');
19  // });
20
21  Route::resource('users',UserController::class)->only([
22      'create','update','show'
23  ]);
```

```
21  Route::resource('users',UserController::class)->except([
22      'update','show'
23  ]);
```

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

Route::resource('users',UserController::class)->names([
    'create' => 'users.build',
    'show' => 'users.view'
]);
```

Laravel : Route for Multiple Resource Controller

Single Resource Controller Route

```php
Route::resource('users', UserController::class);
```

Multiple Resource Controller Route

```php
Route::resource([
    'users' => UserController::class,
    'post' => PostController::class
]);
```

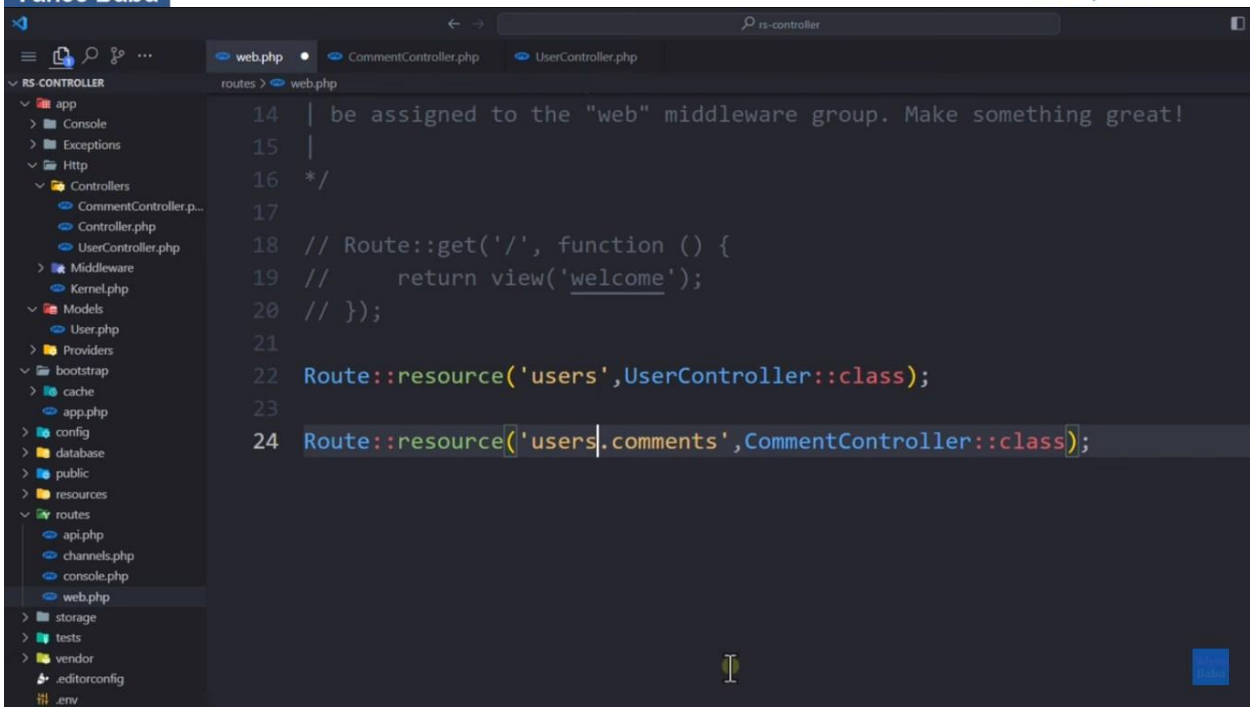
Laravel : Nested Resources

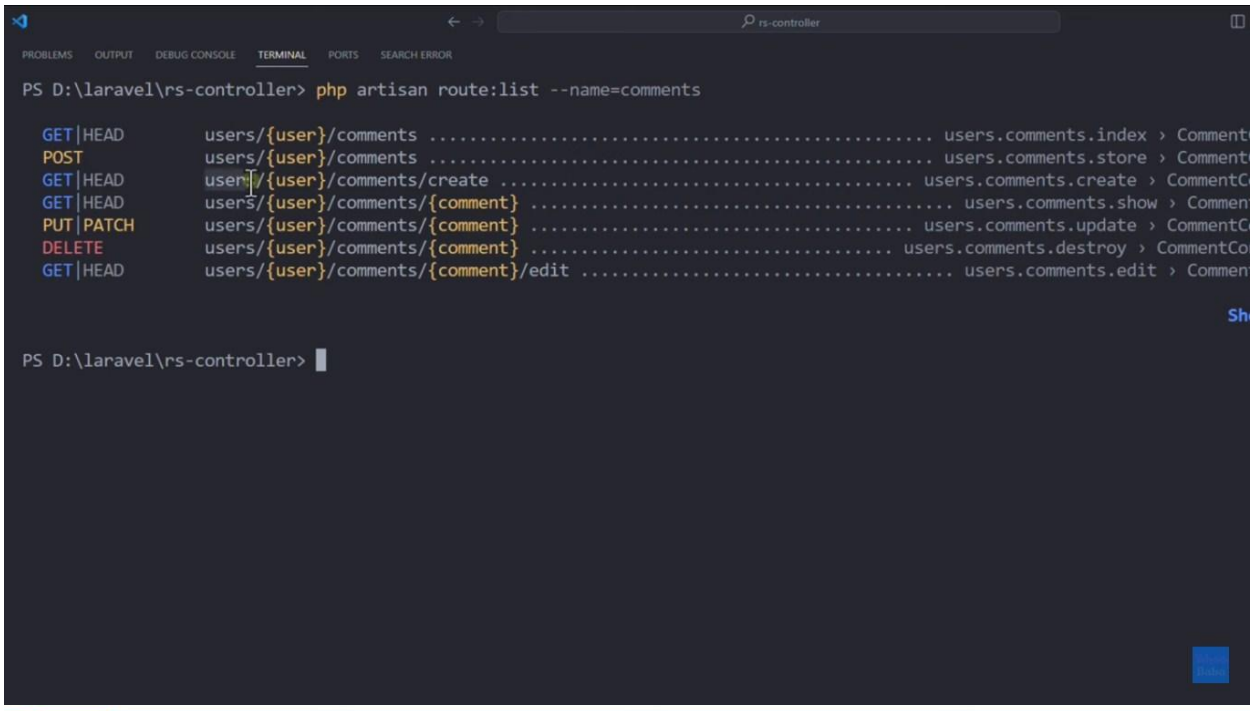
routes/web.php

Users → Comments

```
use App\Http\Controllers\UserCommentController;

Route::resource('users.comments', UserCommentController ::class);
```

Verb	URI	Action	Route Name
GET	/users/{user}/comments	index	users.comments.index
GET	/users/{user}/comments/create	create	users.comments.create
POST	/users/{user}/comments	store	users.comments.store
GET	/users/{user}/comments/{comment}	show	users.comments.show
GET	/users/{user}/comments/{comment}/edit	edit	users.comments.edit
PUT/PATCH	/users/{user}/comments/{comment}	update	users.comments.update
DELETE	/users/{user}/comments/{comment}	destroy	users.comments.destroy





Laravel : Join Table

Students Table

Id	Name	Age	City
1	Ram Kumar	19	1
2	Salman Khan	18	3
3	Meera Khan	19	2
4	Sarita Kumari	21	1

Cities Table

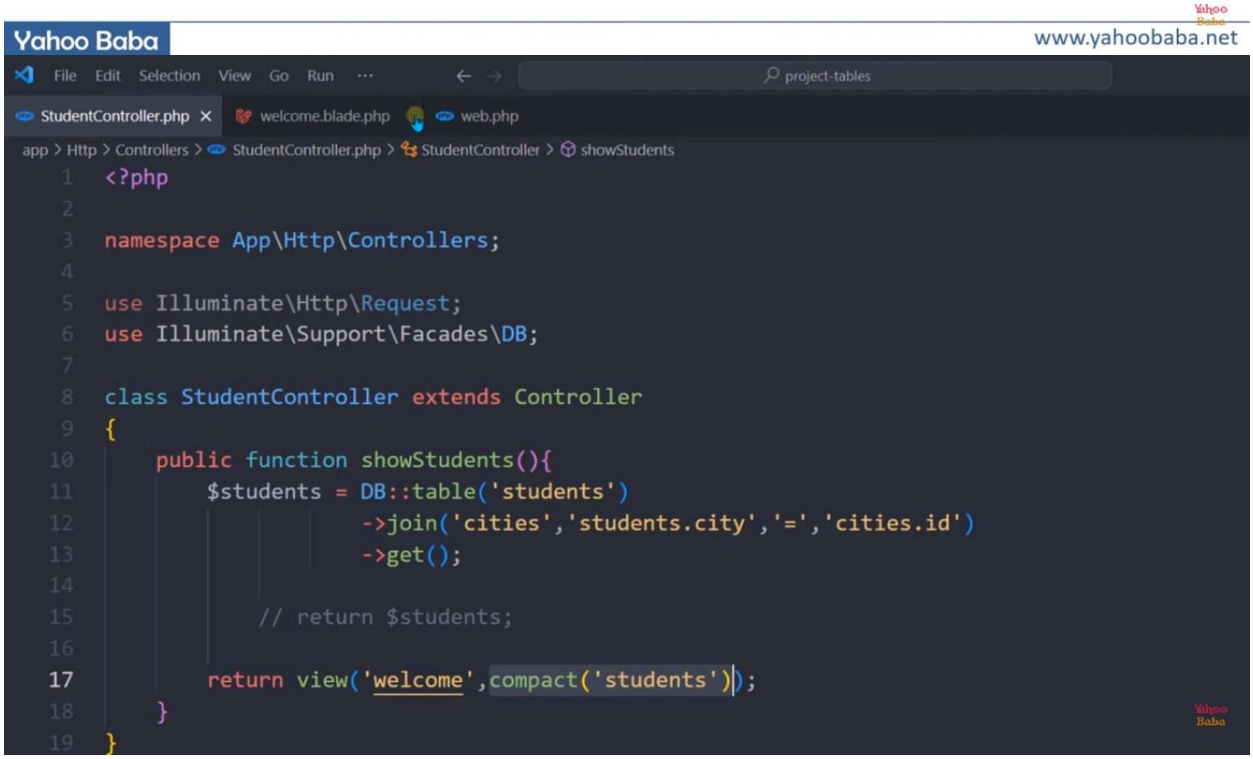
Cid	City
1	Agra
2	Bhopal
3	Delhi

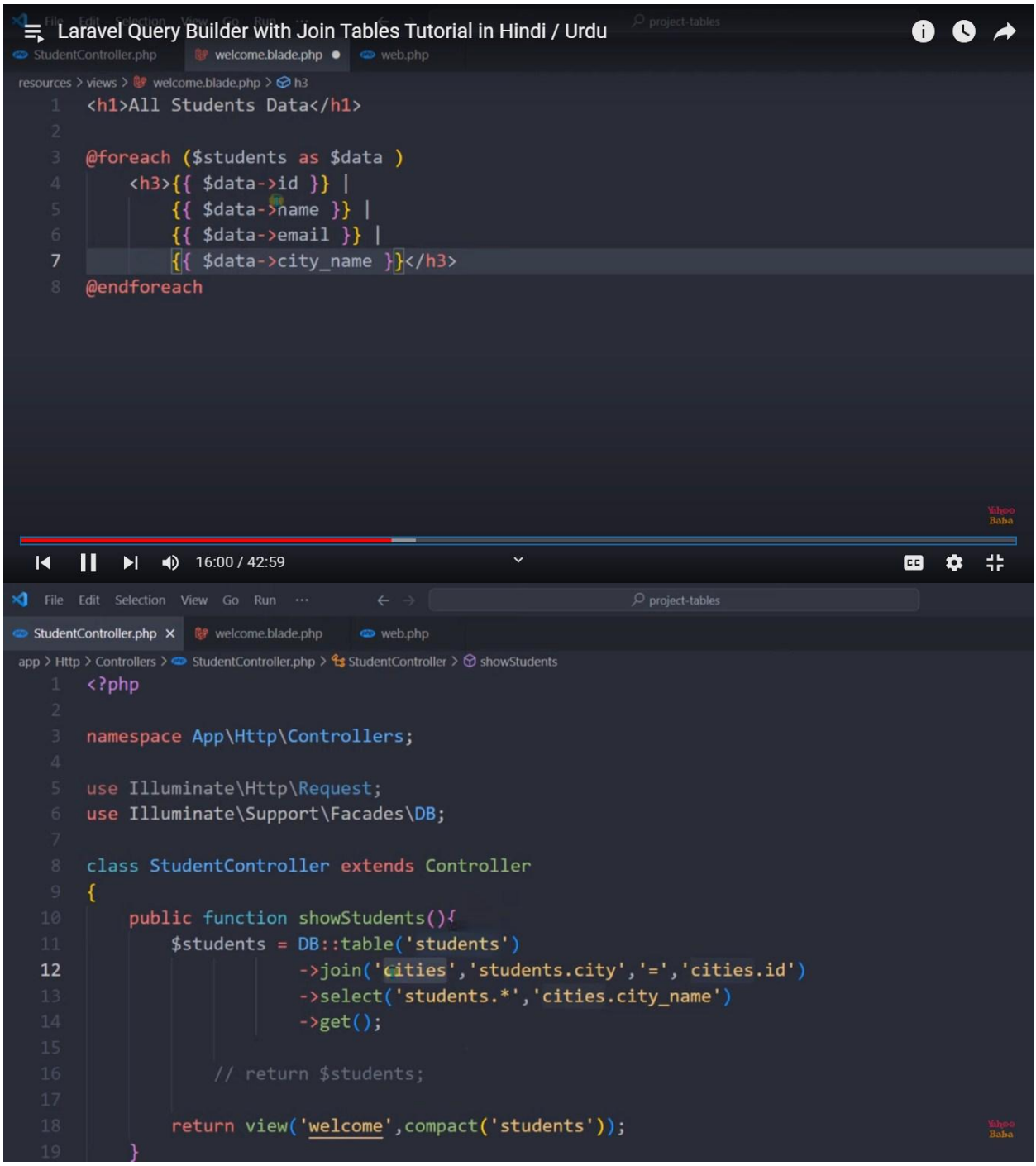
```
SELECT *
FROM students
INNER JOIN cities
ON students.city = cities.cid;
```

```
DB::table('students')
->join('cities','students.city','=','cities.cid')
->get();
```

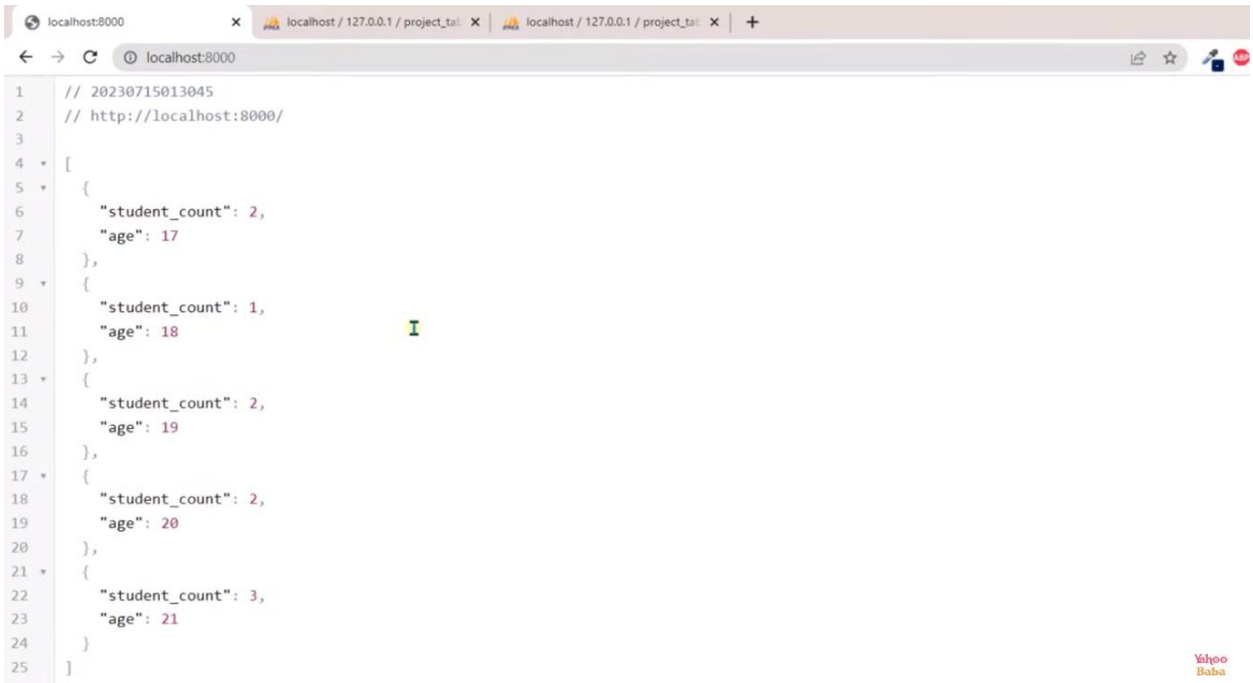
Laravel : MySQL Joins

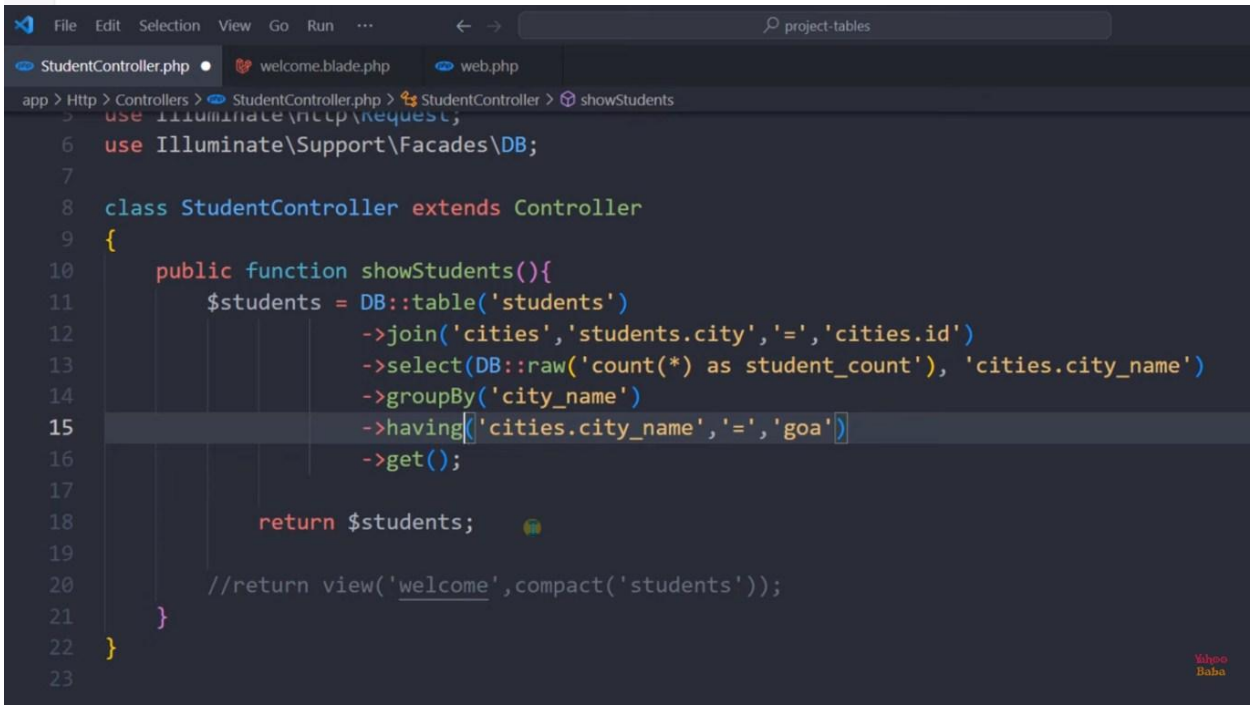
- INNER JOIN `join()`
- LEFT JOIN `leftJoin()`
- RIGHT JOIN `rightJoin()`
- CROSS JOIN `crossJoin()`

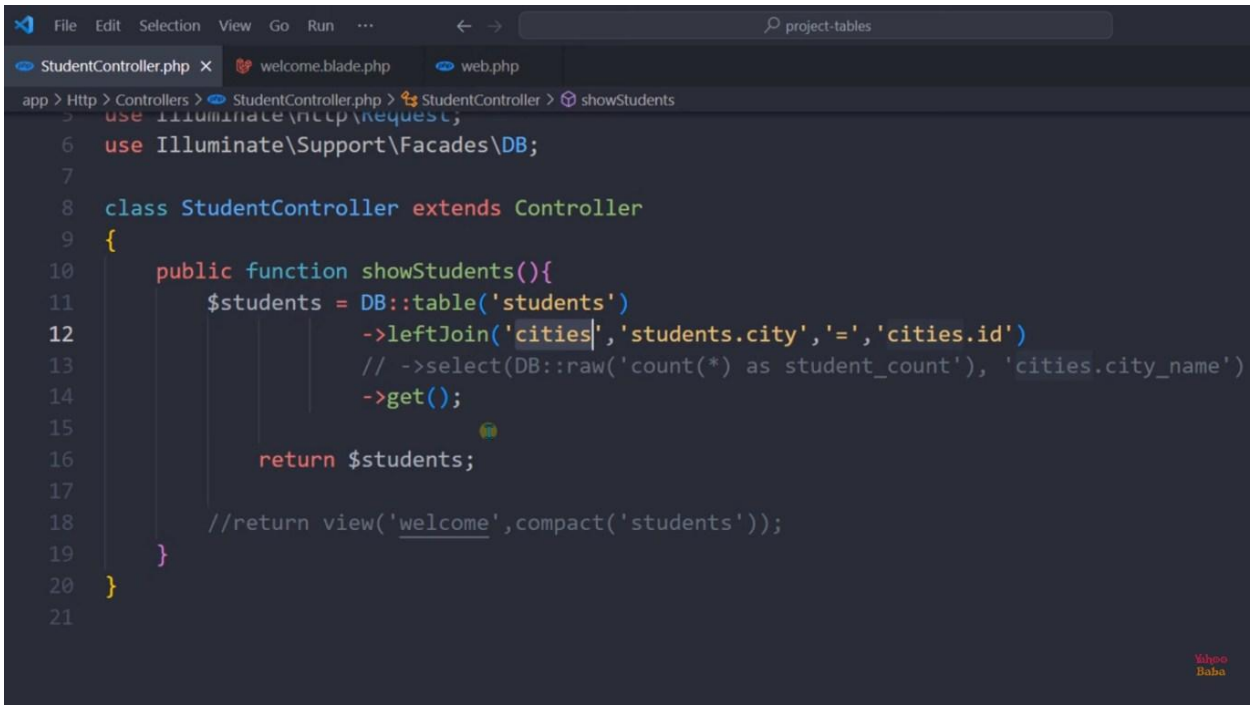



Laravel Query Builder with Join Tables Tutorial in Hindi / Urdu
StudentController.php
welcome.blade.php
web.php
resources > views > welcome.blade.php > h3
<h1>All Students Data</h1>
@foreach ($students as $data)
<h3>{{ $data->id }} |
{{ $data->name }} |
{{ $data->email }} |
{{ $data->city_name }}</h3>
@endforeach
16:00 / 42:59
File Edit Selection View Go Run
project-tables
StudentController.php
welcome.blade.php
web.php
app > Http > Controllers > StudentController.php > StudentController > showStudents
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
class StudentController extends Controller
{
public function showStudents(){
$students = DB::table('students')
->join('cities','students.city','=','cities.id')
->select('students.*','cities.city_name')
->get();
// return $students;
return view('welcome',compact('students'));
}


File Edit Selection View Go Run ···
project-tables
StudentController.php
welcome.blade.php
web.php
app > Http > Controllers > StudentController.php > StudentController > showStudents
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
class StudentController extends Controller
{
public function showStudents(){
$students = DB::table('students')
->join('cities','students.city','=','cities.id')
->select('students.*','cities.city_name')
->where('students.email','=','akshay@gmail.com')
->get();
// return $students;
return view('welcome',compact('students'));
File Edit Selection View Go Run ···
project-tables
StudentController.php
welcome.blade.php
web.php
app > Http > Controllers > StudentController.php > StudentController > showStudents
use Illuminate\Support\Facades\DB;
class StudentController extends Controller
{
public function showStudents(){
$students = DB::table('students')
->join('cities','students.city','=','cities.id')
->select(DB::raw('count(*) as student_count'), 'age')
->groupBy('age')
// ->where('cities.city_name','=','goa')
->get();
return $students;
//return view('welcome',compact('students'));
}
}


localhost:8000
localhost / 127.0.0.1 / project_tal
localhost / 127.0.0.1 / project_tal
localhost:8000
// 20230715013045
// http://localhost:8000/
[
{
"student_count": 2,
"age": 17
},
{
"student_count": 1,
"age": 18
},
{
"student_count": 2,
"age": 19
},
{
"student_count": 2,
"age": 20
},
{
"student_count": 3,
"age": 21
}
]


File Edit Selection View Go Run
project-tables
StudentController.php
welcome.blade.php
web.php
app > Http > Controllers > StudentController.php > StudentController > showStudents
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
class StudentController extends Controller
{
public function showStudents(){
$students = DB::table('students')
->join('cities','students.city','=','cities.id')
->select(DB::raw('count(*) as student_count'), 'cities.city_name')
->groupBy('city_name')
->having('cities.city_name','=','goa')
->get();
return $students;
//return view('welcome',compact('students'));
}
}


File Edit Selection View Go Run ···
project-tables
StudentController.php
welcome.blade.php
web.php
app > Http > Controllers > StudentController.php > StudentController > showStudents
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
class StudentController extends Controller
{
public function showStudents(){
$students = DB::table('students')
->leftJoin('cities','students.city','=','cities.id')
// ->select(DB::raw('count(*) as student_count'), 'cities.city_name')
->get();
return $students;
//return view('welcome',compact('students'));
}
}
Yahoo Baba