**CodeIgniter 4**

**namespaces :**

In CodeIgniter 4, namespaces are used to organize and structure your application's code into logical groups, making it easier to manage and maintain your project. Namespaces help prevent naming conflicts and allow you to autoload classes and files using the PSR-4 autoloading standard.

Here's how namespaces work in CodeIgniter 4:

**Defining Namespaces :**

You define namespaces at the beginning of your PHP classes or files. Typically, you define a namespace that corresponds to the directory structure of your application. For example, if you have a controller located in the "App\Controllers" directory, you would define its namespace like this:

**namespace App\Controllers;**

**Using Namespaces :**

To use a class or file from a different namespace, you need to either provide the fully qualified class name or import the class using the use statement. For example:

**use App\Controllers\HomeController;**

**// Then you can create an instance of HomeController**

**$homeController = new HomeController();**

**Autoloading :**

CodeIgniter 4 supports PSR-4 autoloading, which means that classes are automatically loaded based on their namespace and directory structure. You configure autoloading in the app/Config/Autoload.php file. By default, the framework is set up to autoload classes within the App namespace.

**$psr4 = [**

**'App' => APPPATH, // The 'App' namespace corresponds to the 'app' directory.**

**'CodeIgniter' => SYSTEMPATH, // The 'CodeIgniter' namespace corresponds to the system directory.**

**];**

**Controllers and Models :**

When you create controllers and models in CodeIgniter 4, they should be placed in the appropriate namespace directories, such as App\Controllers and App\Models, respectively.

**Routing :**

When defining routes in CodeIgniter 4, you specify the controller's fully qualified class name (including the namespace) in the routes configuration.

**$routes->get('users', 'App\Controllers\UserController::index');**

By using namespaces in CodeIgniter 4, you can better organize your code, improve code readability, and avoid naming conflicts between classes and functions. This helps create a more maintainable and scalable application.