Unit-7 Creating and Using Composer Packages

- INTRODUCTION:-

- Laravel is a free, open-source PHP web application framework designed for web development. It is known for its elegance, simplicity, and readability.
- Some of the key features of Laravel are its routing system, built-in authentication and authorization, support for MVC architecture, blade templating engine, and more.
- The topics covered in this presentation are related to the advanced features of Laravel, specifically Creating and Using Composer Packages, Ajax and jQuery, and Security & Session.

byteXL

Composer Packages

- Composer is a dependency manager for PHP, which allows us to easily install and manage external packages in our Laravel project
- .
- To install Composer, we need to download and run a Composer installer. After installing Composer, we can use it to install packages from Packagist, which is a repository of PHP packages.

- We can also create our own Composer packages for Laravel, which can be used in other Laravel projects.

- Example: Let's create a simple package that adds a "Hello, World!" message to our Laravel application. First, we create a new Laravel package using the following command:
- **composer create-project --prefer-dist laravel/laravel hello-world-package**

byteXL

Then, we create a new file named HelloWorld.php in the src folder of our package. Here's the code for HelloWorld.php:

```
<?php

namespace HelloWorld;

class HelloWorld
{
    public function getMessage()
    {
        return "Hello, World!";
    }
}
```

- Finally, we add the package to our Laravel project by updating the composer.json file:

```
{
  "require": {
    "hello-world-package/hello-world": "dev-master"
  },
  "repositories": [
    {
      "type": "path",
      "url": "../hello-world-package"
    }
  ]
}
```

We can now use the HelloWorld class in our Laravel application by adding the following code to our controller:

byteXL

```
use HelloWorld\HelloWorld;

public function index()
{
   $helloWorld = new HelloWorld();
   $message = $helloWorld->getMessage();

   return view('welcome', ['message' =>
$message]);
}
```

byteXL

Using Ajax and jQuery

- Ajax (Asynchronous JavaScript and XML) is a technique for creating fast and dynamic web pages. It allows us to send and receive data from the server asynchronously, without requiring a full page reload.
- jQuery is a JavaScript library that simplifies the process of working with HTML documents, handling events, and making Ajax requests.
- We can use Ajax and jQuery in Laravel to create dynamic web pages and add interactivity to our application.
- Example: Let's create a simple search functionality in our Laravel application using Ajax and jQuery. First, we create a new route in our web.php file:

Default Route Files

```
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'index']);
```

Routes defined in the routes/api.php file are nested within a route group by the RouteServiceProvider. Within this group, the /api URI prefix is automatically applied so you do not need to manually apply it to every route in the file. You may modify the prefix and other route group options by modifying your RouteServiceProvider class.

byteXL

Building Rest API

REFER 3rd experiment in Laboratory paper

byteXL

Building a RESTful API with routes Eloquent ORM

- Refer LAB - 2

byteXL

Eloquent ORM Models,

- Laravel includes Eloquent, an object-relational mapper (ORM) that makes it enjoyable to interact with your database.
- When using Eloquent, each database table has a corresponding "Model" that is used to interact with that table.
- In addition to retrieving records from the database table, Eloquent models allow you to insert, update, and delete records from the table as well.

byteXL

Naming conventions

- Models generated by the make:model command will be placed in the app/Models directory. Let's examine a basic model class and discuss some of Eloquent's key conventions:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model
{
    //
}
```

Table Names

- After glancing at the example above, you may have noticed that we did not tell Eloquent which database table corresponds to our Flight model. By convention, the "snake case", plural name of the class will be used as the table name unless another name is explicitly specified. So, in this case, Eloquent will assume the Flight model stores records in the flights table, while an AirTrafficController model would store records in an air_traffic_controllers table.

- If your model's corresponding database table does not fit this convention, you may manually specify the model's table name by defining a table property on the model:

Table Names

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model
{
    /**
     * The table associated with the model.
     *
     * @var string
     */
    protected $table = 'my_flights';
}
```

byteXL

Primary Key

- Eloquent will also assume that each model's corresponding database table has a primary key column named id. If necessary, you may define a protected $primaryKey property on your model to specify a different column that serves as your model's primary key:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model
{
    /**
     * The primary key associated with the table.
     *
     * @var string
     */
    protected $primaryKey = 'flight_id';
}
```

byteXL

TimeStamps

- By default, Eloquent expects created_at and updated_at columns to exist on your model's corresponding database table. Eloquent will automatically set these column's values when models are created or updated. If you do not want these columns to be automatically managed by Eloquent, you should define a $timestamps property on your model with a value of false:

TimeStamps

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model
{
    /**
     * Indicates if the model should be timestamped.
     *
     * @var bool
     */
    public $timestamps = false;
}
```

byteXL

Thank You

byteXL