Unit-6 Using Controllers and Routes for URLs and APIs

- INTRODUCTION:-

- Instead of defining all of your request handling logic as closures in your route files, you may wish to organize this behavior using "controller" classes.
- Controllers can group related request handling logic into a single class.
- For example, a UserController class might handle all incoming requests related to users, including showing, creating, updating, and deleting users. By default, controllers are stored in the app/Http/Controllers directory.

byteXL

Creating a basic controller

- **Let's take a look at an example of a basic controller. Note that the controller extends the base controller class included with Laravel: App\Http\Controllers\Controller:**

```
<?php

namespace App\Http\Controllers;

use App\Models\User;

class UserController extends Controller
{
    /**
     * Show the profile for a given user.
     *
     * @param  int  $id
     * @return \Illuminate\View\View
     */
```

byteXL

Creating a basic controller

```
public function show($id)
    {
        return
view('user.profile', [
            'user' =>
User::findOrFail($id)
        ]);
    }
}
```

byteXL

Creating a basic controller

- **You can define a route to this controller method like so:**

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

Route::get('/user/{id}', [UserController::class,
'show']);
```

byteXL

Creating a route using a closure

- **The most basic Laravel routes accept a URI and a closure, providing a very simple and expressive method of defining routes and behavior without complicated routing configuration files:**

```
use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {
    return 'Hello World';
});
```

byteXL

Default Route Files

- All Laravel routes are defined in your route files, which are located in the routes directory. These files are automatically loaded by your application's App\Providers\RouteServiceProvider.
- The routes/web.php file defines routes that are for your web interface. These routes are assigned the web middleware group, which provides features like session state and CSRF protection.
- The routes in routes/api.php are stateless and are assigned the api middleware group.

- For most applications, you will begin by defining routes in your routes/web.php file. The routes defined in routes/web.php may be accessed by entering the defined route's URL in your browser.
- For example, you may access the following route by navigating to http://example.com/user in your browser:

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
{
    //
}
```

byteXL

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';
}
```

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