AWT
MST

Name : Raushan kumar
CRN : 2221139
URN : 2203751

Q1. Evaluate the following statement :
"As developer. I should use Bootstrap".

ans ⇒ The statement "As a developer, I should use Bootstrap can be evaluated based on several factors:

Advantages / Pros of using Bootstrap:

① Rapid Development: Bootstrap provides pre-built components, styles and Javascript plugins, that helps rapid development process.

② Cross-Browser Compatibility : It ensures consistent rendering across different browsers and devices.

③ Responsive Design: Bootstrap is designed to adapt to various screen sizes, making it suitable for mobile-first development.

④ Large Community and Support: Bootstrap has a vast community of developers actively contributes to Bootstrap, providing extensive resources and support.

Disadvantages / Cons of using Bootstrap

① Learning Curve: ~~While Bootstrap simplifies~~ Even though Bootstrap makes some ~~thing~~ things easier, but you still need to spend time to learning how to use it and it's own syntax.

(ii) Customization Limitations: Bootstrap can be customized ~~but~~ but it might not always fit your exact design forcing to make adjustment.

(iii) Bloat: Using the whole Bootstrap library can add extra, unused code, slowing down the site.

(d) Dependency: Relying too much on Bootstrap can make it difficult to manage or modify your website independently, without depending on the framework.

Conclusion: Whether or not I should use Bootstrap depends on our project requirements. if I need quick, responsive, well-tested UI components with minimal setup, Bootstrap is a good choice.

However, for highly custom or lightweight projects, I should not use Bootstrap.

Q2. Differentiate between git status and git diff command.

Features	git status	git diff
purpose	It shows the current state of your working directory.	It shows the differences between various states of your files.
Shows/ output	(i) Lists files that are ready for commit. (ii) Shows files that have changed but are not yet commit. (iii) Shows untracked files.	(i) Shows line-by-line change in that file that are not yet commit. (ii) Shows chemges between the working directory and the latest commit. (iii) useful for seeing exactly what changes have been made in modified files.
Helps	Helps in determining which files have been modified, added, or deleted before Commetting.	Helps in reviewing the actual content changes before committing.
command	Basic: git status	Basic: git diff staged files: git diff --stageo
common use case	checking the overall status before committing	Reviewing detailed changes before committing.

Q3. How the grid system is effective for creating tables? Show with the help of an example.

ans⇒ The grid system is effective for creating tables because it provides a flexible and responsive way to arrange content. This approach is commonly used in CSS frameworks like Bootstrap, which utilize a grid system to align elements in rows and columns.

Ex:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title> Grid system Table Example</title>
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
bootstrap@5.0.2/dist/css/bootstrap.min.css">
</head><body>
<div class="container mt-5">
  <div class="row">
    <div class="col-md-12">
      <table class="table table-bordered">
        <thead>
          <tr>
            <th>Name</th>
            <th>Age</th>
            <th>Email</th>
          </tr>
        </thead>
        <tbody>
        <tbody>
          <tr>
            <td>John Doe</td>
            <td>30</td>
            <td>John.doe@example.com</td>
          </tr>
          <tr>
            <td>Jane smith</td>
            <td>25</td>
```

```
<td>Jane.smith@example.com</td>
</tr>
<tr>
    <td>Bob Johnson</td>
    <td>40</td>
    <td>bob.Johnson@example.com</td>
</tr>
</tbody>
</table>
</div>
</div></div>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

(pyQf) pushing & pulling from remote repo.

(Q4) You have made several commits in your local repository Elaborate the process of pushing and pulling these changes to and from a remote repository. include the necessary Git commands and steps

ans → ✱ Pushing changes to a remote repository

Step 1: Stage changes:

git add <file1> <file2> .... # Add specific files

or for pushing all files write

git add . # Add all files of the current directory

Step 2: Commit changes: Commit the staged changes with a descriptive message.

git commit -m "commit message"

Step 3: push changes: push commits to the remote repository

git push origin <branch-name> # typically main or # master branch.

Pulling changes from a Remote Repository

① Step 1: Fetch changes: Fetch the changes from the remote repository to update your local repository with any new changes:

"git fetch origin" # this command retrieves & updates but doesn't merge them into local branch.

Step 2: Merge changes: To integrate the fetched changes into your local branch use

git merge origin <branch-name> # replace <branch-name> you want to merge changes such as 'main' or 'master'

Alternatively, you can use git pull which combines git fetch and git merge into a single command

"git pull origin <branch-name>"

Step 1: "git init" initialize git in your project

Step 2: Add a Remote Repository: Add remote repository url by running the following command.

• git remote add origin https://github.com/CoderRaythem

Q5. Create a Bootstrap-based navigation bar for a website that includes dropdown menues.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title> nav bar </title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css">
</head>
<body>
<nav class="navbar navbar-expand-lg navbar-light">
<a class="navbar-brand href="#">Brand</a>
<button class="navbar-toggler type="button"
data-bs-toggle="collapse" data-bs-target="#navbarNav">
<span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarNav">
  <ul class="navbar-nav">
    <li class="nav-item"><a class="nav-link" href="#">
      Home</a></li>
    <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#"
    id="servicesDropdown" role="button" data-bs-toggle=
"dropdown">Services</a>
  <ul class="dropdown-menu" aria-labelledby="
  serviceDropdown">
  <li><a class="dropdown-item" href="#">
    web Dev</a></li>
```

```
<li><a class="dropdown-item" href="#">
  SEO</a></li>
<li><a class="dropdown-item" href="#">
  DSA
</a></li>
<li><hr class="dropdown-divider"></li>
<li><a class="dropdown-item" href="#">
  Marketing</a></li>
</ul>
</li>
<li class="nav-item"><a class="nav-link"
  href="#">contact</a></li>
</ul>
</div>
</nav>
<script src="https://cdn.jsdelivr.net/npm/bootstrap
@5.0.2/dist/js/bootstrap.bundle.min.js">
</script>
</body>
</html>
```

Q6. Imagine you are working on a collaborative web development project using Bootstrap 5 and Git. Your team is facing challenges in maintaining consistent styling and layout across different web pages. Additionally, you need to address related to version control conflicts in Git.

(a) Propose a strategy, for ensuring consistent styling and layout across multiple web pages using Bootstrap 5.

(b) Outline a plan to manage version control conflicts effectively in Git, considering the challenge faced by your team. Above 2 points must be addressed with the help of student registration

for admission to any program.

(a) Ensuring consistent styling and layout with Bootstrap 5.

(i) Centralized CSS and Javascript files: create a central style.css file where you define all custom styles. Link this file to all your web pages. Similarly, have a central script.js file for any custom Javascript.

(2.) Utilize Bootstrap's Grid System: Use Bootstrap's grid system to create a consistent layout. For example, use containers, row, and columns to structure your pages. This ensures that all pages follow the same layout principles.

(3) Bootstrap components: Use Bootstrap components like forms, buttons and cards to maintain a uniform look. for example use Bootstrap's form classes for the registration form to ensure consistency.

(4) Custom Bootstrap Theme: Customize Bootstrap using Sass to create a theme that matches your projects branding. This ensures that all Bootstrap components have a consistent look and feel.

(5) Reusable components: Create reusable components for common elements like headers, footers, and navigation bars, This ensures that these elements look the same across all pages.

~~(b) Managing Version Control Conflicts in Git.~~

Student registration form

PYQF this question

```
<!DOCTYPE html>
<html lang="en">
<head>
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css">
<title> bootstrap form </title>
</head>
<body>
<div class="container mt-5">
<h1 class="text-center"> Bootstrap form </h1>
<div class="row">
<form action="/registration" method="post">
<div class="col-6 offset-3 mb-3">
<label for="name"> Name </label>
<input type="text" class="form-control"
id="name" placeholder="Enter your Name">
</div>
<div class="col col-6 offset-3 mb-3">
<label for="email"> Email </label>
<input type="email" class="form-control" id="email" placeholder="Enter your Email">
</div>
<div class="col col-6 offset-3 mb-3">
<label for="password"> password </label>
<input type="password" class="form-control"
id="address" placeholder="Enter your Address">
</div>
<div class="col col-6 offset-3 mb-3">
<label for="state"> State </label>
<select id="state" class="form-control">
```

```
<option selected>Choose to Select State</option>
<option>.Bihar</option>
<option>punjab</option>
<option>Delhi</option>
</Select>
</div>
</div class="col col-6 offset-3 mb-3">
<button type="submit" class="btn btn-warning">
sign in </button>
</div>
</form>
</div>
</div>
</body>
</html>
```

(Q5) ~~How does Git handle conflicts during the merging of branch~~

(b) Managing Version control conflicts in Git.

(i) Branching strategy: Implement a branching strategy like Git flow or feature work-flow. Each team member works on their own branch and ~~merge~~ merges changes into the main branch only after testing the code.

```
" git checkout -b feature/student-registration"
```

(ii) Regular commits and pulls: Encourage team members to commit their changes regularly and pull the latest changes from the main branch frequently to minimize conflicts.

```
" git add ."
" git commit -m " Commit message"
" git pull origin main"
```

③ Code Reviews and pull Requests : Use pull requests for merging changes into the main branch. This allows for code reviews and ensures that conflicts are resolved before merging.

```
"git checkout main"
"git pull origin main"
"git merge feature/student-registration"
```

④ ~~Conflic~~ write Descriptive commit messages : write meaningful commit messages that explain the changes made. This helps other team members understand the purpose of the changes and avoid conflicts.

"git commit -m "changing bg color""

⑤ Conflict Resolution Tools and Techniques : Use Git's built-in conflict resolution tools (eg ⇒ git mergetool) to visually compare and merge conflicts changes.

---

⑤ what is git, advantages of git and why do we use it.

Git is a distributed version control system that allows multiple developers to work on a project simultaneously by tracking changes in files. It helps in managing the development of software projects by storing different ~~vers~~ versions of files.

Advantages of git :

① Version control : Git tracks changes in your code, allowing you to ~~previous v~~ revert to previous version of code.

② collaboration : Multiple developers can work on the same project simultaneously without conflicts, as git manages changes made by different team members.

③ Branching: Git allows developers to create separate branches for features, bug fixes, or experiments. These branches can later be merged to the main branch.

④ Backup: Since the entire repository is stored on multiple machines (local and remote), Git acts as a distributed backup system for your projects.

⑤ Efficient and fast: Git is optimized for performance and operations like committing, branching and merging are very fast.

⑥ Open Source and free: Git is free to use and has strong community support.

Why we use Git

(i) To track changes: Developers use git to keep a record of every modification made to the code.

(ii) Collaboration: It allows multiple developers to work together on a project.

(iii) Branching and Merging: Developers can test and develop features separately without effecting the main project. and can be merged when well tested.

(iv) Distributed System: Developers have a complete copy of the repository, making it easier to work offline and ensure redundancy.

8 Marks: Design a codeigniter / Laravel web application to demonstrate model, view and controller for crud operation for a 'user' table, take user date of birth and display it's age, month, delta, soft delete concept, log table, routing, crud. (12 marks pyq 1 time specific question CI)

project structure: ① Model : Handle database interactions
② View : Displays data to the user.
③ Controller : Manages user requests and response

Advantage of MVC: The Model view controller - pattern offers several benefits.

① Separation of concerns: Divide application logic (Model), UI (view) and user input (controller), making each part independent.

② Easier Maintenance: Components can be modified individually enhancing scalability.

③ Better collaboration: Different teams can work on Model, View, or Controller separately.

④ Code Reusability: Logic and UI elements can be reused.

⑤ Facilitates Testing: Each part can be tested independently, making bug detection easier.

⑥ Enhanced control: Provides structured control over application behavior and user interactions.

~~1. Database Structure:~~

what is MVC: MVC is a software design pattern used to organize and structure code in applications, it is a like frame work in web development, it divides an application into three interconnected components.

① Model ② View ③ Controller

1. Database structure

```
CREATE TABLE User (
id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(50),
email VARCHAR(50),
date_of_birth DATE,
is_deleted TINYINT(1) DEFAULT 0, --0: active, 1: soft delete
create_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE log (
log_id INT AUTO_INCREMENT PRIMARY KEY,
action VARCHAR(50),
user_id INT,
timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. Routing in routes.php (Routing)

```
$route['user'] = 'UserController/index'; // show all user
$route['user/add'] = 'UserController/create'; // Add user
$route['user/edit/(:num)'] = 'UserController/edit/$1'; // Edit
$route['user/delete/(:num)'] = 'UserController/soft_delete/$1'; // Soft delete
```

3. Controller/(UserController.php)

```
class UserController extends CI_Controller {
    public function __construct() {
        parent::__construct();
        $this->load->model('UserModel');
    }
    public function index() {
        $data['users'] = $this->UserModel->get_users();
        $this->load->view('user_view', $data);
    }
```

```
public function create(){
    $user_data = [
    'name' => 'John Doe',
    'email' => 'john@example.com',
    'date_of_birth' => '1990-01-01'
    ];
    $this->UserModel->addUser($user_data);
}
public function edit($id){
    $user_data = ['name' => 'Updated Name'
    ];
    $this->UserModel->update_user($id, $user_data);
}
public function soft_delete($id)
{   $this->UserModel->soft_delete_user($id);
}
}
```

4. Model/(UserModel.php)

```
class UserModel extends CI_Model{
  public function get_users(){
    $this->db->where('is_deleted', 0);
    $query = $this->db->get('user');
    return $query->result();
  }
public function add_user($data){
    $this->db->insert('user', $data);
    $this->log_action('create', $this->db->insert_id());
}
public function update_user($id, $data)
{   $this->db->where('id', $id);
    $this->db->update('user', $data);
    $this->log_action('update', $id);
}
```

```
Public function soft_delete_user($id){
    $this->db->where('id', $id);
    $this->db->update('user', ['is_deleted' => 1]);
    $this->log_action('soft_delete', $id);
}

Private function log_action($action, $user_id){
    $this->db->insert('log', [
        'action' => $action,
        'user_id' => $user_id
    ]);
}
}
```

5. view/user_view.php

```
<!Doctype html>
<html>
<head>
    <title> User List </title>
</head>
<body>
<h2> Users List </h2>
<table>
    <tr>
        <th> Name </th>
        <th> Email </th>
        <th> Age </th>
    </tr>
<?php foreach ($users as $user): ?>
    <tr>
        <td><?= $user->name ?></td>
        <td><?= $user-> email ?></td>
        <td><?php
            $dob=new DateTime($user->date_of_birth);
            $now=new DateTime();
            $interval=$now->diff($dob);
```

```
echo $interval->y . "Years" . $interval->m . "Months" .
    $interval->d . "days";
    ?>
  </td>
 </tr>
<?php endforeach; ?>
</table>
</body>
</html>
```

Q Definition of Laravel and codeigniter its advantages and disadvantages and diff b/w laravel and codeigniter

Laravel: Laravel is a robust, open-source PHP framework used for developing web applications, known for its elegant syntax, scalability, and a wide range of built-in tools like Eloquent ORM, Blade templating engine, and Laravel Artisan CLI.

Codeigniter: Codeigniter is a lightweight and simple PHP framework designed for developers who need to straightforward toolkit for building dynamic websites with minimal overhead.

Advantage of Laravel:

(i) Rich Ecosystem and Tools: Laravel provides a wide range of built-in features such as Eloquent ORM, Blade templating engine, and laravel mix.

(ii) Security: offers strong security features like CSRF protection, XSS protection, encrypting and password hashing.

(iii) Routing and Middleware: Advanced routing capabilities, middleware support for handling HTTP requests.

(iv) Database Handling: Laravel's Eloquent ORM makes database interaction easier and more readable. It also supports migration,

Sending, and multiple database connections.

(5) Artisan CLI: The command-line interface allows for automation of ~~test~~ repetitive tasks like database migration, seeding and artisan commands.

Disadvantages of Laravel:

(i) Performance: Laravel can be slower than lightweight frameworks due to its rich features and build-in libraries.

(ii) Learning curve: Laravel's wide range of features can make it more difficult for beginners to learn.

(iii) Complexity for small projects: Laravel might be overkill for small projects due to its complexity and heavy dependencies.

(iv) ~~Frequent~~ Requires more server resources: Laravel can be resource-intensive, requiring more memory and CPU.

Advantages of CodeIgniter.

(1) Simple and Lightweight: CodeIgniter is lightweight and straightforward, making it easy to get started with and ideal for smaller Applications.

(2) Easy to use: It is easy to learn and implement specially for beigner developers.

(3) Performance: CI is optimized for performance, making it faster and less resource-heavy.

(4) Good for small Applications: It is an excellent choice for small to medium-sized applications.

(5) Documentation: Codeigniter has good documentation, making it easier for developers to understand and use the framework.

Disadvantage of CI

(i) Limited features: CodeIgniter comes with fewer built-in features than Laravel.

(ii) NO ORM: CodeIgniter does not provide an object-Relational Mapping (ORM) system like Laravel's Eloquent.

(iii) Security: While CodeIgniter provides some security features but it doesn't provide that much security features like laravel.

(iv) Limited support for Modern features: CI lacks support for some modern web development features like injection, middleware, advance routing.

(v) Smaller Community: Compared to laravel codeigniter has a smaller community.

features	Laravel	CodeIgniter
1. Architecture	follows MVC with more advanced features.	Simple MVC, for small Applications.
2. Database interaction	It uses Eloquent ORM with built-in database migrations.	It uses Active Record for database interactions, no ORM
Templates	Blade templating engine with advantage features.	No built-in templating engine, uses pain php.
Routing	Advanced routing, supports resource routing, route groups and middleware	Simple and easy routing system
Authentication	Built-in authentication system with guard support.	Requires manual setup for authentication.
Performance	slightly slower	faster
Learning curve	difficult for beginners	Easy for beginners.

(Q9) Explain database migration and benefits of database migration in Laravel.

ans → Database migration is just like version control system for your database. It helps you create and update tables and columns in structured way, using commands instead of manually editing the database.

Ex → Migration file to create users

["php artisan make:migration users"]

```
public function up()
{
    Schema::create('users', function(Blueprint $table)
    {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamps();
    });
}
public function down()
{
    Schema::dropIfExists('users');
}
```

Benefits of database migration

(i) version control for database: Tracks changes to the database schema over time.

(ii) Easy for collaboration: Developers can share migration files in version control systems like Git.

(iii) Automated schema management: Avoids manual database modification

(iv) Role Back changes: If a schema changes ~~mistakenly~~ by mistake it allows you to roll back to previous versio

Q. Explain Eloquent ORM: Eloquent ORM (object-relational mapping) in laravel is a way to interact with your database using models, making database queries simple instead of writing raw SQL. It provide different methods for create, update, delete and read user data.

Ex⇒

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

// Eloquent methods

```
use App\Models\User;
$users = User::all();
```

Get all data

// find and update

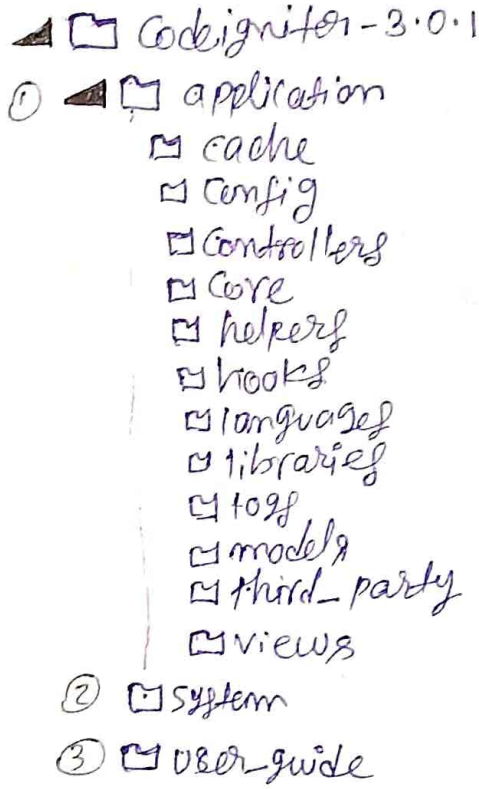
```
$user = User::find(1);
$user->name = 'up';
$user->save();
```

* Concept of Routing in Laravel and Code igniter:

Ans→ Routing in Laravel and Codeigniter is the process of how an application should respond to a specific HTTP requests. In laravel routes are defined in the 'routes' 'web.php' file and 'routes/api.php'. In codeigniter routes are defined in 'application/config/routes.php'. Each route is associated with a specific URI, and a controller method. such as returning a view or executing a controller method.

Q In CodeIgniter, describe the directory structure and ~~Codei~~ configurations.

folder structure ⇒



CHLMV

When you open the directory structure of codeigniter there will main three folders which are given:

① Application ② System ③ User_guide

Application: The application folder is where all the code of the app we are developing is stored. It consists several other folders.

(i) Cache: In this folder, all the cache pages of App will be stored.

(ii) Config: In this folder, all the configuration files are stored

(iii) Controllers: In this folder, it contains the control of Application and all server-side functionalities.

(iv) Core: All the base classes of application will be stored.

(v) Helpers: This will help you to ~~modify the inner working of your framework~~ increase creating your application.

(v) Hooks: This will help you to modify the inner working of framework

(6) languages: you can use the language according to your need.

(7) logs: Here all the files related to the log will store.

(8) Models: All the database logins will be stored here.

(9) Third_party: All the third_party pluging will be stored here.

(10) views: Here your all HTML files related to the applications will be stored.

- System: All the files related to coding, libraries, and other files will be stored here. This folder also contains various folders.

(1) Core: It consists of all the Codeigniter's core classes.

(2) database: All the drivers and utilities regarding ~~fonts are stored here~~. database will store here.

(3) fonts: All the information and utilities regarding fonts are stored here.

(4) Helpers: It consists of all helpers-related data such as date, cookie etc.

(5) languages: All language-related files stored here codeigniter supports multilingual web app.

(6) ~~Library~~: Libraries: Here libraries will be stored.

User_guide: It works as an offline codeigniter guide which helps you to learn the basic functions of various libraries of Codeigniter. It consists of an index.php file that contains important things to set environmental and error level.

- CodeIgniter configuration files

(i) index.php: This file is the entry point of the application, located in the root directory.

(ii) config.php: found in application/config, this file allows you to set configurations like the base_URL, encryption keys, and error logging.

(iii) database.php: This file stores database connections settings. including hostname, database name, username and password.

(iv) autoload.php: Lets you define which libraries helpers, and models should be loaded automatically.

(v) routes.php: Configures URL routing, including the default controller and custom routes.

(1) Discuss the concept of routing with example in the context of Laravel framework.

ans ⇒ Routing in Laravel is the process of defining how an application should respond to specific HTTP requests. In Laravel, routes are defined in the routes/web.php file and routes/api.php: Each route is associated with a specific URI (Uniform Resource Identifier) and an action, such as returning a view or executing a controller method.

TYPES of Routing in Laravel

(1) Basic Routing (2) Route parameters (3) Named Routes (4) Route Grouping (5) Controller Routing

1. Basic Routing: Defines a simple route to return a string or view.

Ex⇒

```
Route::get('/hello', function(){
    return "Hello, world!";
});
```

② Route parameters: Allow dynamic values in the URI.

Ex⇒

```
Route::get('/user/{id}', function($id){
    return "User ID:". $id;
});
```

③ Named Routes: Give routes a name, making it easier to refer to them later in code.

Ex⇒

```
Route::get('/dashboard', function(){
    return view('dashboard');
})->name('dashboard');
```

④ Route Grouping: Organizes routes with shared attributes, like middleware or prefixes.

Ex⇒

```
Route::prefix('admin')->group(function(){
    Route::get('/users', function(){
        return "Admin User List";
    });
});
```

⑤ Controller Routing: Directs a route to a specific controller and method.

Ex⇒

```
Route::get('/profile', [UserController::class, 'showProfile']);
```

Example: Consider a basic example where we want to create routes for showing a user profile and editing it.

```
Use App\Http\Controllers\UserController;
Route::get('/user/{id}', [UserController::class, 'show']);//show profile
Route::get('/user/{id}/edit', [UserController::class, 'edit']); // edit profile
```

Q. Create a laravel web application with route definition that can be redirected to the web application home page.

Ans ⇒ To create a basic laravel web application with a route that redirects to the home page, follow these steps:

① Install Laravel: First, create a new Laravel project

```
laravel new myApp
```

② Define the Home Route: Open the routes/web.php file and add a route for the home page.

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

This route listens for the '/' URL and returns the welcome view, which is the default home page for Laravel application

③ Redirect Route to Home: Add a route that redirects any other route to the home page.

```
Route::get('/home', function(){
    return redirect('/');
});
```

Here, visiting /home will redirect the user to the root (/) route, showing the home page.

Example Code for routes/web.php

```
<?php
use Illuminate\Support\Facades\Route;
Route::get('/', function(){
    return view('welcome');
});
Route::get('/home', function(){
    return redirect('/');
});
// This setup ensures that visiting either / or /home will show home page.
```

Q Analyze the following statement with respect to any PHP framework: "Use URLs based on the requirement instead of using the predefined URLs"

ans ⇒ The statement refers to the ability to define custom routes or URLs that fit the specific needs for the application, rather than relying on the the default or predefined URLs.

"url: 'profile/{id}'" instead of using a generic or predefined URL like "user/1"

Most PHP frameworks like Laravel or Codeigniter allows developers to create custom routes using a routing system. This means developers can specify URL patterns that align with the business logic or user-experience, rather than set of predefined URLs.

Ex⇒ Laravel:

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

Assignment - 2

Name : Raushan Kumar
CRN : 2221139
URN : 2203751

Q1. Discuss the concept of routing with example in the context of Codeigniter framework?

In the Codeigniter framework, routing is the process of defining URL patterns that map to specific controllers and their methods, determining what should happen when a user visits a particular URL. This helps to create clean, user-friendly URLs and organize how requests are handled within the applications.

How Routing works in Codeigniter.

When a URL is accessed, Codeigniter's routing system checks the URL pattern against the routes defined in the routes.php file (located in the app/config directory). If a match is found Codeigniter loads the specified controller and method, otherwise it shows a 404 error page.

Route Syntax in Codeigniter:

```
$routes->get('url-pattern', 'controllerName::methodName');
```

URL pattern: The part of the URL after the base URL.

controllerName::methodName: Specifies the controller and method that should handle this route.

Types of Routes in Codeigniter

① Default Route: Specifies the default controller and method to load when no other route matches.

```
$routes->get('/', 'Home::index'); // Load the 'index' method of the 'Home' Controller.
```

② Custom Route: maps to a specific URL pattern to a controller method.

```
$routes->get('about', 'Pages::about');
```

③ Wildcard Routes: Allows dynamic segments in the URL pattern to capture. → Route with parameter.

```
$routes->get('product/(:num)', 'product::view/$1');
        //map → /product/123 to view
```

④ HTTP verb-specific Routes: Codeigniter supports routing based on HTTP methods (GET, POST, PUT, DELETE).

```
$routes->post('submit-form', 'FormController::submit');
    //only accessible via a POST request.
```

⑤ 404 Override: customizes the 404 page when no route matches.

```
$routes->set404Override('Errors::show404');
// calls the 'show404' method
```

Examples of Routing in Codeigniter

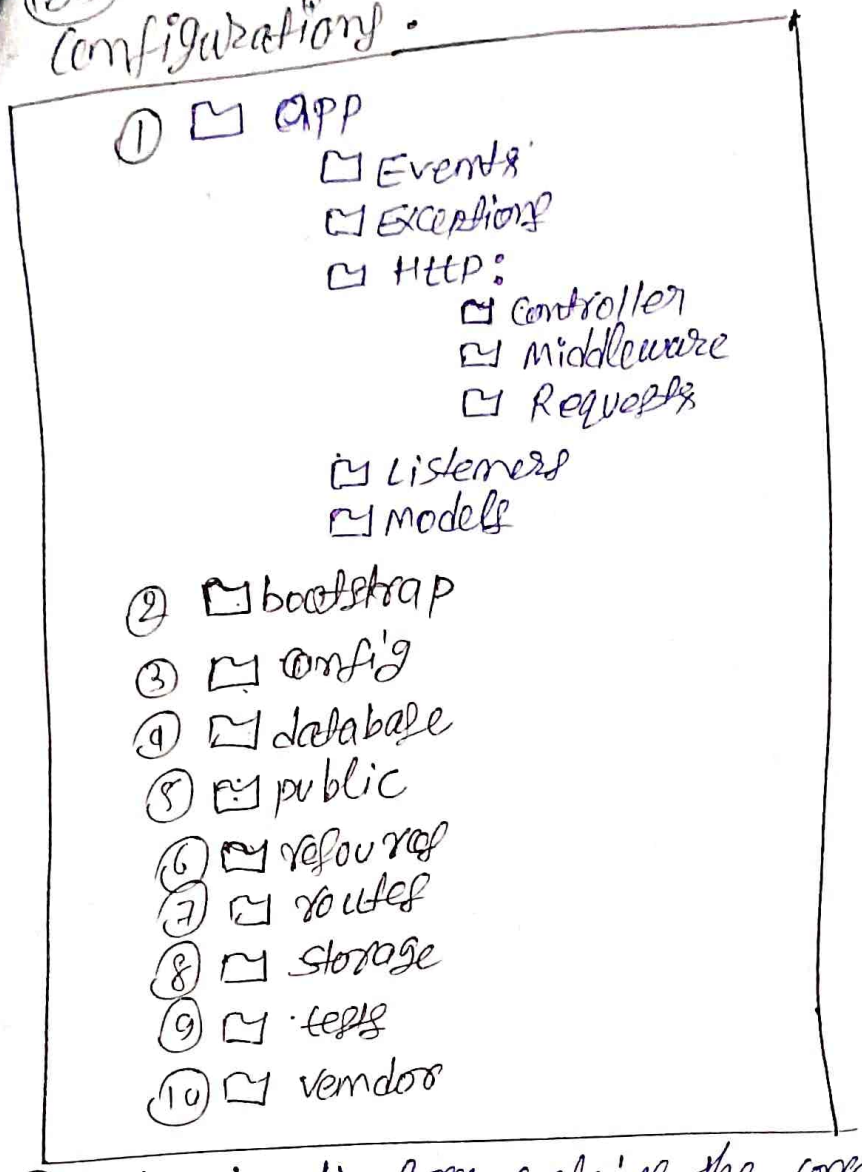
Suppose we have a website where we want to route different pages such as 'home', 'about', and 'product details' to specific controllers and methods.

1. Define the routes in routes.php

```
// default route
$routes->get('/', 'Home::index'); // Home page route
// Static page route → About
$routes->get('about', 'Pages::about');
// Dynamic route for products with ID as a parameter
// Product detail page
$routes->get('product/(:num)', 'Product::view/$1');
```

enough

(Q.8) In Laravel, describe the directory structure and configurations.



abcdprrstv

(1.) app/: This directory contains the core code of the application.
- Events: Defines application events.
- Exceptions: Contains the Handler.php for handling exceptions.
- Listeners: Defines Event Listeners.
- Models: Contains Eloquent models.
- Http: (a) Controller: Houses application controllers.
  (b) Middleware: Defines HTTP middlewares.
  (c) Requests: Stores custom request validation logic.

(2) bootstrap/: Contains the framework bootstrap file and cache files for configuration, services and packages.
eg: app.php

③ config/: Holds all configuration files for the appli.
eg → app.php, database.php, queue.php, view.php

④ Database: containing database-related files
Eg → migrations: Holds migration files

⑤ public/: The entry point for the application
Eg ⇒ index.php

⑥ resources/: stores fronted assets and views:
Eg ⇒ views → Contains Blade templates
css/js → Holds raw frontend assets
lang ⇒ stores language files.

⑦ routes: Defines application routes
- web.php: Routes for web interface
- api.php: Routes for APIs.
- console.php: Routes for console based commands.

⑧ storage/: Stores logs, compiled templates, and file uploads

⑨ tests: containing automated tests
- feature/: feature tests.
- unit: unit tests.

⑩ vendor/: Containing all composer dependencies and packages.

---

configurations ① Environment variables (.env): stores credentials informations. Eg → db credentials, port, URI

② config/app.php: Key settings like application name etc.

③ config/database.php: Database connections settings (MD, SQL)

④ config/mail.php: Email server configuration.

⑤ config/cache.php: Cache settings etc.

⑥ config/session.php: Session management settings.

(Ayo) (12m) Design a web application or to demonstrate laravel model, view, routes and controller with CRUD.

Ans ⇒

Step1: Install laravel:

```
Composer create-project laravel/laravel StudentCrud
```

Step2: Create a migration for Students:

```
php artisan make:migration Students
```

Step3: Update migration file:

```
public function up()
{ Schema::create('Students', function(Blueprint $table){
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->string('phone');
    $table->timestamps();
}});
```

Step4: Run the migration:

```
php artisan migrate
```

Step5: Model : Create a model

```
php artisan make:model Student
```

Step6: Update Student.php

```
protected $fillable=['id','name','email','phone'];
```

Step7: Controller : Create Controller

```
php artisan make:Controller StudentController
```

Step8: Implements crud methods in controller (StudentController).

```
use App\Models\Student;
use Illuminate\Http\Request;
class StudentController extends Controller
{
    // display all data of student
    public function index()
    {
        $students = Student::all();
        return view('students.index', compact('students'));
    }
    // show the form to create a new student
    public function create()
    {
        return view('students.create');
    }
    // store a new student
    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required',
            'email' => 'required|email|unique:students',
            'phone' => 'required',
        ]);
        Student::create($request->all());
        return redirect()->route('students.index');
    }
    // show the form to edit student info
    public function edit(Student $student)
    {
        return view('students.edit', compact('student'));
    }
    // update a student
    public function update(Request $request, Student $student)
    {
        $request->validate([
            'name' => 'required',
            'email' => 'required|email|unique:students,email,' . $student->id,
            'Phone' => 'required', ]);
```

```
        $student->update($request->all());
        return redirect()->route('students.index');
    }

    // delete a student
    public function destroy(Student $student)
    {
        $student->delete();
        return redirect()->route('students.index');
    }
}
```

Step 9: Route: Defines Routes: Add routes in routes/web.php:

```
use App\Http\Controllers\StudentController;
Route::resource('students', StudentController::class);
```

Step 10: Views: Create views (resources/views/students)

// 1. index view: resources/views/students/index.blade.php

```
<h1>Student List</h1>
<a href="{{ route('students.create') }}">Add user</a>
<table>
    <tr>
        <th>Name</th>
        <th>Email</th>
        <th>phone</th>
        <th>Actions</th>
    </tr>
@foreach($students as $student)
    <tr>
        <td>{{ $student->name }}</td>
        <td>{{ $student->email }}</td>
        <td>{{ $student->phone }}</td>
```

```
<td>
<a href="{{ route('students.edit', $student->id) }}">
Edit
</a>
<form action="{{ route('students.destroy', $student->id) }}"
method="POST">
@csrf
@method('DELETE')
<button type="submit">Delete</button>
</form>
</td>
@endforeach
</table>
```

2. Create a student : resources/views/students/create.blade.php

```
<h1>Add student</h1>
<form action="{{ route('students.store') }}" method="POST">
@csrf
<label>Name:</label>
<input type="text" name="name">
<label>Email:</label>
<input type="email" name="email">
<label>phone:</label>
<input type="text" name="phone">
<button type="submit">Save</button>
</form>
```

3. Edit a student : resources/views/students/edit.blade.php

```
<h1>Edit Student</h1>
<form action="{{ route('students.update', $student->id) }}"
method="post">
@csrf
@method('PUT')
<label>Name:</label>
<input type="text" name="name" value="{{ $student->
name }}">
```

```
<label>Email:</label>
<input type="email" name="email"
value="{{ $student->email }}">
<label>Phone:</label>
<input type="text" name="phone"
value="{{ $student->phone }}">
<button type="submit">update</button>
</form>
```

Step 11: Start the server

```
php artisan serve
```

Navbar with dropdown bootstrap code

```
<div class="container">
  <h2 class="text-center">Bootstrap navbar</h2>
  <div class="row">
    <div class="col">
      <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand">Company</a>
        <ul class="navbar-nav">
          <li class="nav-item">
            <a class="nav-link">Home</a>
          </li>
          <li class="nav-item">
            <a class="nav-link">Gallary</a>
          </li>
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" data-toggle="dropdown">
              About Us
            </a>
            <div class="dropdown-menu">
              <a class="dropdown-item">Link1</a>
              <a class="dropdown-item">Link2</a>
              <a class="dropdown-item">Link3</a>
            </div>
          </li>
        </ul>
        <form class="form-inline">
          <input type="search" class="form-control" placeholder="search">
          <button class="btn btn-warning">Search</button>
        </form>
      </nav>
    </div>
  </div>
</div>
```

drop down

(diff) · Local Git and remote Git

features	Local Gist	remote Gist
Definition	Local gist refers to the Gist repository stored on your local machine.	remote Gist refers to the gist repository hosted on remote server (eg Github)
Internet connection	Does not require an internet connections.	Requires an internet connection.
Accessibility	Accessible only on the local machine.	Accessible globally to users
Data storage	stores repository data in [.gist] folder locally	Stores repository data on a remote server
Team collaboration	No collaboration possible	team collboration
Backup	No backups	Acts as a backup of the repo
Security	More Secure	less Secure
Commands	git add., git commit, git branch	git pull, git fetch, git merge
Example	A repo you initialized with "git init"	A repo hosted on ~~Team~~ "github".

ITBSCode

features	Bootstrap 4	Bootstrap 5
jquery	it requires jQuery	it doesn't required
custom forms	custom forms are available but with a limited customization	it provides more customizable forms
Cards	cards with limited style and options	it has more styles with additional options
icon library	it does not include icon library.	it introduces its own icon library

Disha: Believe me that much content will be more than enough for AWT.