Architecture Overview

The final project for this course involves creating a web application using **Laravel** with **MySQL**, **Docker**, **CI/CD (GitLab)**, and other modern tools and practices. Below is the high-level breakdown of the project, including steps, technologies, and interactions between components.

Project Breakdown:

Fork the GitHub Repository (Template Repository):

- **Fork** the repository to your GitHub account.
- **Clone** the repository into your local or Cloud IDE environment.
- The main web application is a predefined **Laravel application**. We will need to **add new pages** and build the necessary features for the project implementation.

Set Up the Docker Application:

- **Install Docker** with the necessary packages for containerization.
- **Run the project** to verify whether it's working or not within the Docker environment.
- **Create static pages** to meet the user story requirements (e.g., landing page, about page, etc.).

Run and Test the Application Locally:

- Test the **local development environment** to ensure the application is running as expected.
- Ensure that the **Docker containers** (Laravel, MySQL, etc.) are set up correctly and can communicate with each other.

Add User Management:

- Implement **Laravel's built-in authentication system** for **Login** and **Registration**.
- Use **HTML/CSS/JS** with **Bootstrap** to implement frontend views for user management, including login and registration forms.

Create Database and Backend Services:

- Set up **MySQL** database to store necessary data, including:
  - **Car models**, **Dealers**, and **Reviews**.
- Create **Laravel models** for **Car Make**, **Car Model**, **Dealers**, and **Reviews**.
- Implement **CRUD operations** for managing cars, dealers, and reviews in the backend.

6. **Integrate External Services (Optional)**:

- Integrate **Sentiment Analysis API** (using an external service like **IBM Watson** or a custom PHP-based solution).
- Implement **RESTful APIs** to interact with external services (e.g., fetching dealer details, analyzing review sentiment).

7. **Create Dynamic Pages and UI**:

- Use **Blade templates** in Laravel to create dynamic pages, including:
  - A page to display **All Dealers**.
  - A page to display **Reviews for a selected dealer**.
  - A page to **Add a review** for a selected dealer.
- Implement **frontend validation** using **JavaScript** and **Bootstrap** for forms.

8. **CI/CD Pipeline with GitLab**:

- Set up a **GitLab CI/CD pipeline** to automate the following processes:
  - **Code linting** (using tools like **PHP CodeSniffer** or **PHPStan**).
  - **Unit testing** with **PHPUnit**.
  - **Building** and **deploying Docker containers** for both the Laravel app and the MySQL database.

9. **Deploy the Application**:

    - Deploy the **Dockerized Laravel application** on a **Kubernetes cluster** (or Docker Swarm).
    - Set up a **reverse proxy** using **Nginx** (or Apache) to handle incoming requests and direct them to the appropriate containers.
    - Deploy the **MySQL database** either:
        - Inside a Docker container.
        - Or, use an **external MySQL service**.

Solution Architecture Overview

User Interaction:

1. **Frontend (HTML, CSS, Bootstrap, JS)**:

    - The user interacts with the **Dealership Website** via a browser.

- Users can:
  - View **dealer listings**.
  - View **reviews** for dealers.
  - **Add reviews** for a selected dealer.

2. **Laravel Application**:
   - The **Laravel Application** serves as the backend, processing requests and managing the application logic through:
     - **Routes** and **controllers** for API requests like:
       - `/cars` – Fetch a list of cars.
       - `/dealers` – Fetch a list of all dealers.
       - `/dealers/{state}` – Fetch dealers based on a state.
       - `/dealer/{id}` – Fetch details of a specific dealer.
       - `/reviews/dealer/{id}` – Fetch reviews for a dealer.
       - `/add-review` – Post a review for a dealer.
3. **Database**:
   - **MySQL** database stores:
     - **Car Makes** and **Car Models**.
     - **Dealer information**.
     - **Reviews** (including sentiment data).
   - **Laravel's Eloquent ORM** is used for seamless database interaction.
4. **Sentiment Analysis Service(optional)**:
   - An external **Sentiment Analysis API** is integrated into the Laravel application through HTTP requests.
   - Reviews are sent to the sentiment analysis service to determine whether they are **positive**, **negative**, or **neutral**.
5. **Dockerized Services**:
   - **Laravel Application** runs within a **Docker container**, including:
     - **PHP**, **Composer**, **Nginx**.
     - **MySQL** (either as a Docker container or an external service).

- **Docker Compose** is used to manage multiple containers (e.g., Laravel, MySQL, Nginx).

6. **CI/CD with GitLab**:
   - **GitLab CI/CD** automates the following:
     - **Code linting** and **unit testing** (PHP CodeSniffer, PHPUnit).
     - **Build Docker images** for both the Laravel app and MySQL.
     - **Push the Docker images** to a Docker registry (e.g., Docker Hub).
     - **Deploy the application** to a **Kubernetes cluster** or **Docker Swarm**.

Detailed Flow and Microservices

1. **Frontend Service (Laravel + Blade)**:
   - The **frontend** is created using **Laravel's Blade** templating engine:
     - Static pages like **home page** and **contact page**.
     - Dynamic pages like:
       - **List of dealers**.
       - **Reviews for a selected dealer**.
       - **Add reviews** for dealers.
2. **Backend Services (Laravel Controllers)**:
   - Laravel handles API requests such as:
     - `/dealers` – Get a list of dealers.
     - `/dealers/{state}` – Get dealers based on state.
     - `/dealer/{id}` – Get specific dealer details.
     - `/reviews/{dealer_id}` – Get reviews for a dealer.
     - `/add-review` – Handle review submissions.
   - It also communicates with external services like the Sentiment Analyzer API.
3. **Database (MySQL)**:
   - Stores data for:
     - **Car Make** and **Car Models**.

        - **Dealers**.
        - **Reviews**, including the **sentiment** analysis result.

4. **Sentiment Analyzer**:
    - The Sentiment Analysis Service provides an endpoint:
        - `/analyze/:text` – Returns **positive**, **negative**, or **neutral** sentiment for a review.

5. **Dockerized Laravel Application**:
    - The **Dockerfile** is used to:
        - Install **PHP**, **Composer**, **Nginx**, and **Laravel dependencies**.
        - Set up **MySQL** container.
        - Expose the Laravel application on a specific port.

6. **GitLab CI/CD Pipeline**:
    - The **GitLab CI/CD** pipeline automates:
        - Running **tests** (PHPUnit).
        - **Linting** the PHP code (PHPStan, PHP CodeSniffer).
        - **Building** the Docker image and pushing it to the Docker registry.
        - **Deploying** the containerized application to a Kubernetes or Docker Swarm cluster.

Deployment

1. **Local Development**:
    - Set up a **local development environment** using Docker and Docker Compose.
    - Test the **Laravel application**, **MySQL**, and other dependencies locally.

2. **Production Deployment**:
    - **Push Docker images** to a Docker registry.
    - **Deploy** the application to a **Kubernetes cluster** using **Helm charts** or manual configuration.
    - Set up a **reverse proxy** (Nginx or Apache) for production.
    - **Manage environment variables** and secrets for production settings.

Technologies and Tools:

- **Backend**: Laravel, PHP, MySQL.
- **Frontend**: HTML, CSS, JavaScript, Bootstrap.
- **Containerization**: Docker, Docker Compose.
- **CI/CD**: GitLab CI/CD pipeline.
- **Sentiment Analysis**: External API (IBM Watson or custom).
- **Deployment**: Kubernetes (Dockerized app), Docker Hub.
- **Version Control**: GitLab, GitHub.