PREMIER UNIVERSITY, CHATTOGRAM

Department of Computer Science and Engineering

Report

Course Code	**:** CSE 338
Course Title	**:** Software Development
Report No.	**:** 01
Report Title	**:** Contact Management Application
Date of Submission	**:** 23/10/2024

Submitted By:

Name: Sukanta Das
ID: 1502910200969
Semester: 6th
Section: d
Batch: 29th
Session: Spring 2024

Remarks:

Report on Contact Management Application

1. Introduction

The Contact Management Application is designed to streamline the organization and management of personal and business contacts. Users can easily create, view, edit, and delete contact details, such as names, phone numbers, and email addresses. The application follows the Model-View-Controller (MVC) architectural pattern, ensuring clean separation between the interface, business logic, and data management.

2. Directory Structure Overview

a. app/

Http/Controllers/ContactController.php: Manages the business logic for contact operations, handling the complete lifecycle of a contact (Create, Read, Update, Delete).

- index(): Lists all contacts.

- create(): Displays a form to add new contacts.

- store(): Handles form submissions to add a new contact to the database.

- edit(): Retrieves an existing contact for editing.

- update(): Saves changes to an existing contact.

- destroy(): Deletes a contact from the database.

Http/Requests/ContactRequest.php: Ensures that the user's inputs are validated before they are processed. This helps maintain data integrity by enforcing rules, such as mandatory fields and correct formatting for phone numbers and email addresses.

Models/Contact.php: Represents the data model for contacts. This model maps to a database table and handles all data interactions involving contacts, such as saving, updating, or deleting entries.

b. resources/views/

Blade templates that define the user interface of the application:

contacts/create.blade.php: A form where users can input details to create a new contact.

contacts/edit.blade.php: A pre-filled form for editing an existing contact’s details.

contacts/index.blade.php: A list displaying all contacts.

contacts/show.blade.php: Displays the details of a specific contact.

layouts/app.blade.php: The master layout used across all views, containing headers, footers, and common sections for uniformity.

c. public/

This folder contains the static assets used for the front-end:

custom.css: Defines the styling for the application’s user interface, ensuring an intuitive and visually appealing design.

d. routes/web.php

This file manages routing and URL mapping. It links specific URLs to controller actions, defining how the user interacts with the application. Common routes include:

GET `/contacts`: Retrieves and displays all contacts.

POST `/contacts`: Submits new contact data.

GET `/contacts/{id}/edit`: Loads an existing contact for editing.

DELETE `/contacts/{id}`: Deletes a specific contact.

3. Application Features

The Contact Management Application offers a rich set of features to manage personal and business contacts effectively:

Create Contacts: Users can add new contacts by filling out a form that captures essential information such as name, phone number, and email. The application ensures that the data entered is valid using server-side validation.

View Contacts: A comprehensive dashboard lists all saved contacts with options to view details or navigate to other actions.

Edit Contacts: Allows the user to modify existing contact information. This feature is useful when contact details need to be updated.

Delete Contacts: Contacts that are no longer needed can be permanently removed from the system.

Search and Filter: The app provides search functionality, enabling users to quickly find specific contacts based on criteria like name or email.

4. Technology Stack

The application is built using the following technologies:

Laravel Framework: A PHP framework that powers the back-end and provides tools for routing, database management, and handling requests.

Blade Templating Engine: Laravel’s templating engine, used for rendering dynamic content within views while maintaining reusable components.

MySQL or SQLite: A relational database for managing and storing contact information.

HTML/CSS/JavaScript: Front-end technologies used for structuring the application's user interface, styling the views, and implementing dynamic behaviors such as form validation or AJAX requests.

5. Architecture and MVC Pattern

The Contact Management Application is built on the Model-View-Controller (MVC) architecture, which promotes separation of concerns:

Model: Represents the application's data. The Contact.php model handles interactions with the database.

View: The Blade templates in resources/views/ render the front-end interface with dynamic data.

Controller: The ContactController.php handles the application logic, processing user requests and responses, and interacting with the model to retrieve or store data.

This structure enhances the modularity and maintainability of the application.

6. Challenges and Enhancements

Validation and Security: The use of ContactRequest.php ensures the application maintains data integrity and reduces the risk of malicious inputs.

User Experience: The interface can be improved by adding AJAX functionality to enable real-time data updates, reducing page reloads and improving performance.

API Integration: As a future enhancement, the application could integrate with external APIs like Google Contacts or other contact services to import/export contact data, adding more value to users managing large datasets.

Conclusion

The Contact Management Application is a powerful and user-friendly tool for managing contacts efficiently. With a clean separation of logic using the MVC architecture, it ensures that data, views, and control logic are well-organized, making the application scalable and maintainable. The application provides essential features like contact creation, editing, deletion, and search functionalities, making it suitable for both personal and business use. Future enhancements could include real-time updates and API integration to further enhance functionality and user experience.