National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

Project Report

SE-314 Software Construction

Name	CMS Id	Class	Section
Ayesha Siddiqa	407198	SE-13	B
Navaal Iqbal	409977	SE-13	B
Amna Ahmed	408099	SE-13	B
Abdul Rehman Shahid	412430	SE-13	B

Introduction

The Bonsai Pet Shop project is designed to provide an online platform for purchasing pets (cats and dogs) and DIY pet kits. The system is developed using Laravel, leveraging its MVC architecture to ensure scalability, security, and maintainability.

Phase 1: System Design

Framework

- **Backend:** Laravel (MVC Pattern)
- **Frontend:** Blade templates with optional Vue.js/React components for interactivity
- **Database:** MySQL
- **Hosting:** local host

System Components

1. **Client Side:**
    - User interfaces for Admin, Registered Users, and Guests
    - Dynamic content using JavaScript and CSS
2. **Server Side:**
    - Laravel controllers and middleware for request handling
    - RESTful APIs for external integration
3. **Database Layer:**
    - Relational database to store and manage data

2. Core Features

Pet Management

- Add, update, delete, and display pet details.
- Attributes: Name, Breed, Age, Gender, Price, Description, and Images.

DIY Pet Kit Management

- Add, update, delete, and display DIY pet kits.
- Attributes: Name, Description, Price, Included Items, and Images.

Feedback/Reviews

- Registered users can leave reviews for pets or kits.
- Admins can moderate reviews.

Authentication and Authorization

- User registration and login (email verification).
- Role-based access control for Admins, Registered Users, and Guests.

3. User Roles and Responsibilities

Admin

- Manage pets and DIY kits.

Guests

- Browse pets and kits.

4. User Interface Design

Home Page

- Overview of the shop with featured pets and kits.
- Navigation bar with links to About, Gallery, Services, Products, and Contact Us.

About Page

- Detailed information about the Bonsai Pet Shop, its mission, and values.

Gallery Page

- A visual showcase of available pets, kits, and shop highlights.
- Grid layout with high-quality images.

Services Page

- List of additional services provided by the shop (e.g., pet grooming, training, or consultation).

Products Page

- Catalog of all pets and DIY kits.
- Grid layout with images, prices, and quick actions (Add to Cart, View Details).

Contact Us Page

- Contact form for inquiries.
- Shop location, phone number, and email address

5. Database Design

Tables and Attributes

1. **Users:**
   - ID, Name, Email, Password, Role, Created At, Updated At.
2. **Products:**
   - ID, Name, Category (Pet/DIY Kit), Description, Price, Images, Created At, Updated At.

6. Usability Considerations

- **Responsive Design:** Mobile-friendly interface.
- **Performance Optimization:** Use caching for frequently accessed data.
- **User Feedback:** Provide clear error messages and intuitive navigation.

Phase 2: Implementation and Specification

In this phase, We implemented the core features of the *Bonsai Pet Shop* website using Laravel, a modern PHP framework that prioritizes simplicity and robustness. The implementation followed a **specification-first approach**, meaning that each class and method was defined with clear objectives and behaviors before coding. This systematic approach facilitated better planning, ensured consistency, and minimized bugs during development.

Core Features Implemented

1. User-Friendly and Beautiful Layout

- **Responsive Design**: The website is designed to provide an optimal viewing experience on devices of all sizes, ensuring mobile, tablet, and desktop compatibility.
- **Modern Aesthetics**: Leveraging CSS frameworks like Bootstrap and custom styles, the layout emphasizes simplicity, elegance, and visual appeal.
- **Dynamic Homepage**:
  - Includes engaging sections like *Welcome Message*, *Services*, *Gallery*, and *Contact Us*.
  - Promotes the shop's offerings with intuitive call-to-action buttons and a clean navigation bar.

2. Admin Panel for Inventory and Cart Management

The admin panel is a robust system designed to simplify inventory management and order processing.

- **Add Items to Inventory**:
  - Admins can create new product listings with fields like name, price, category, description, and image uploads.
  - Validation ensures no incomplete data is stored in the database.
- **Manage Inventory (Update/Delete)**:

  - Products can be edited or deleted seamlessly through the admin panel.
  - Confirmation prompts reduce accidental deletions.
- **Cart Management**:
  - Admins can manage customer carts by adding or removing items.

3. Customer-Focused Features

- **Product Catalog**:
  - A dedicated section displaying all available products with details such as price, category, and an image preview.
  - Customers can browse items effortlessly with a visually appealing grid layout.
- **Contact Form**:
  - Allows users to submit inquiries or feedback directly to the shop.
  - Includes fields for name, email, and message, with proper input validation.
  - Successful submissions are stored in the database and acknowledged with a confirmation message.
- **Gallery Section**:
  - Showcases shop highlights, products, and pets through an interactive gallery.
  - Includes lightbox functionality for better image viewing.
- **Services Section**:
  - Clearly outlines the services offered, such as pet grooming, training, and accessories.
  - Designed to inform users and promote engagement.

4. Navigation and Accessibility

- **Intuitive Navigation**: A fixed navigation bar ensures users can easily access any section of the website.
- **SEO and Accessibility Features**:
  - Structured HTML and descriptive metadata were used to enhance search engine visibility.
  - ARIA attributes and alt tags improve usability for assistive technologies.

Specification-Driven Development

Predefined Class and Method Specifications

- Each feature and functionality was first outlined in detail to define the expected input, output, and behavior.
  - **Example**:
    - ProductController: Handles CRUD operations for product inventory, including methods like store(), update(), and destroy().
    - ContactFormController: Manages the processing and storage of customer inquiries, ensuring validations like email format and message length.

Validation and Error Handling

- Strong validation rules were implemented for all forms, ensuring data integrity.
- Comprehensive error handling with user-friendly messages was added to address common issues like form submission errors or invalid inputs.

Blade Templates

- Laravel’s Blade templating engine enabled reusable and maintainable views.
- Dynamic content rendering streamlined the integration of database-driven data with front-end designs.

Additional Implementation Highlights

Database Design

- A relational database schema was designed to handle:
  - Products (e.g., name, category, price, stock, description, images).
  - Customer inquiries from the contact form.
  - Admin credentials for secure access.
- Relationships between tables were optimized using Laravel's Eloquent ORM.

Security Features

- **Authentication**: Admin access is secured using Laravel's built-in authentication mechanisms.
- **Input Sanitization**: All user inputs are sanitized to prevent SQL injection and XSS attacks.
- **CSRF Protection**: Token-based validation ensures secure form submissions.

Testing and Debugging During Implementation

- Regular debugging using Laravel's logging features and browser developer tools.
- Basic tests were written to confirm the functionality of critical features, such as product creation and contact form submissions.

This phase resulted in a **fully functional, visually appealing, and user-friendly web application** that meets the project's requirements. The specification-first approach ensured that the implementation was both methodical and scalable, making the *Bonsai Pet Shop* website a solid foundation for further enhancements.

Phase 3:Version Control and Collaboration

The **Bonsai Pet Shop** project provides an online platform for purchasing pets and DIY pet kits. In Phase 3, the project incorporates **version control and collaboration** to enable smooth teamwork, track changes, and resolve conflicts in the codebase effectively. Here's how the features are integrated and applied:

Integration of Version Control Features

1. **Git for Version Control:**
    - The project uses **Git** as the primary version control tool to track all changes to the codebase.
    - Git commands like git add, git commit, git push, and git pull are used to manage and share updates across the team.
2. **Branching Workflow:**
    - A **branch-per-feature** workflow is adopted to allow team members to work independently without interfering with others' code.
    - Example branches:
        - main (stable production branch)
        - feature/pet-management
        - feature/diy-kits-management
        - feature/authentication
3. **Pull Requests (PRs):**
    - Team members submit **pull requests** for merging their branch into the main branch.
    - PRs are reviewed by peers to ensure code quality and adherence to project standards.
4. **Commit Guidelines:**
    - Structured commit messages (e.g., "Added pet management functionality") to improve clarity in version history.

Collaboration and Conflict Resolution

1. **Collaborative Code Editing:**
   - Multiple team members work on the project simultaneously by utilizing branches.
   - Collaboration tools like **GitHub/GitLab** are employed for hosting the repository and enabling real-time interaction.
2. **Merge Conflict Handling:**
   - Conflicts are resolved using Git's conflict resolution tools (e.g., git merge, git rebase).
   - Tools like **VS Code** or **Sourcetree** are used for visual conflict resolution.
3. **Code Review:**
   - Peer reviews for PRs ensure high-quality code and help identify logical errors.
   - Review tools like GitHub's built-in code review feature are used for inline comments and discussions.
4. **CI/CD Integration:**
   - Continuous Integration tools (e.g., **GitHub Actions**) are used to automatically run tests whenever new code is pushed or merged.
   - This ensures no broken code is introduced into the main branch.

Implementation of Collaboration Features in the Bonsai Project

Ensuring Multi-user Code Editing

- The branching system ensures that every team member works independently without interfering with the work of others.
- Real-time collaboration is supported through GitHub/GitLab, where team members can review and comment on each other's code.

Tracking Changes

- Git's history (git log) keeps a detailed record of all changes made by each user.
- Developers can revert to previous versions if needed, using git checkout or git revert.

Conflict Resolution

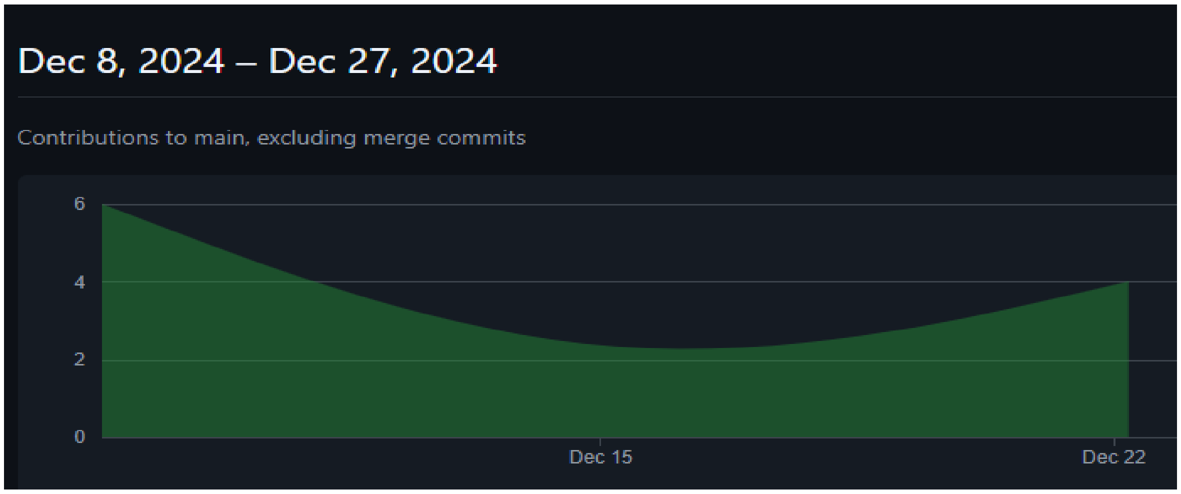
- Merging branches may lead to conflicts. These conflicts are identified by Git and resolved manually or using conflict resolution tools.
- Clear documentation ensures that all team members follow the same coding conventions, minimizing conflicts.

Outcome of Phase 3

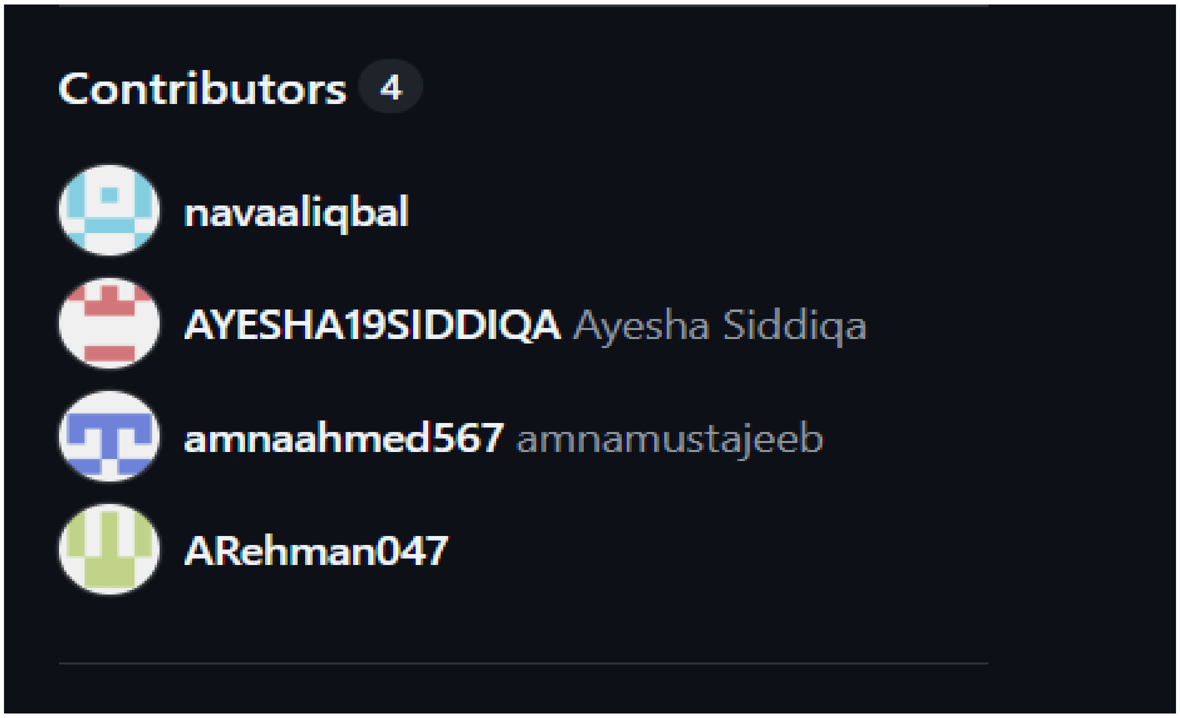
- **Enhanced Team Collaboration:** Developers can work concurrently on different parts of the project.
- **Improved Code Quality:** Regular reviews and CI/CD pipelines ensure a robust and error-free codebase.
- **Efficient Conflict Resolution:** Merging and conflict resolution are handled systematically, reducing downtime.
- **Transparent Versioning:** Every change is tracked, ensuring accountability and clear development history.

By integrating version control and collaboration features, the **Bonsai Pet Shop** project establishes a professional development workflow suitable for a scalable and maintainable application.

Contribution Duration



Contributors



Contributors 4

navaaliqbal

AYESHA19SIDDIQA Ayesha Siddiqa

amnaahmed567 amnamustajeeb

ARehman047

Phase 4: Testing and Validation

Unit and Integration Test Descriptions

Unit Tests

1. ProductControllerTest (Unit Tests)

- **test_create_product_instance**:
  Verifies that a product instance is created with the correct attributes. Ensures the Product model's constructor properly assigns values for category, name, description, and price.
- **test_update_product_instance**:
  Tests the ability to update attributes of a product instance. Confirms the Product model allows modifications to existing attributes such as category, name, description, and price.
- **test_retrieve_product_attributes**:
  Ensures that the attributes of a Product instance can be retrieved correctly. Validates the Product model's getters for all relevant attributes.

Feature Tests

2. ProductControllerTest (Feature Tests)

- **it_displays_all_pet_products**:
  Checks if the index method of the ProductController returns all products to the view. Validates the correct HTTP status, the expected view (products), and the presence of products in the view data.
- **it_stores_a_product**:
  Tests the store method for valid input. Ensures that a product is successfully stored in the database and redirects the user to the products index page.
- **it_validates_product_data_when_storing**:
  Confirms validation rules during product creation. Ensures that incomplete or incorrect data results in validation errors for fields like category, name, and description.
- **it_updates_a_product**:
  Verifies the update functionality. Ensures that the controller updates product attributes in the database and redirects to the index page upon success.
- **it_deletes_a_product**:
  Tests the delete functionality of the ProductController. Ensures that a product is removed from the database and the user is redirected to the index page.

3. HomeControllerTest (Unit Tests)

- **test_auth_middleware_is_applied**:
  Validates that the auth middleware is correctly applied to the HomeController. Ensures unauthenticated users are redirected to the login page.
- **test_index_view_for_authenticated_user**:
  Verifies that the index method returns the correct view (home) for authenticated users.
- **test_unauthenticated_users_redirected_to_login**:
  Confirms that unauthenticated users trying to access the home page are redirected to the login route.
- **test_authenticated_user_can_access_home_page**:
  Ensures that authenticated users can successfully access the home page and the correct view (home) is returned.

4. HomeControllerTest (Feature Tests)

- **test_authenticated_user_can_access_home_page**:
  Ensures that only authenticated users can access the home route. Confirms that the correct HTTP status and view (home) are returned.
- **test_unauthenticated_user_is_redirected_to_login**:
  Verifies that unauthenticated users are redirected to the login page when attempting to access the home route.

5. ContactFormTests (Unit Tests):

- **Email Format Validation:** Ensures the email field accepts only valid email formats, returning errors for invalid inputs.
- **Name Field Validation:** Checks that the name field is mandatory and displays an error if left blank.
- **Message Length Validation:** Verifies that messages meet the minimum length requirement, returning errors for overly short inputs.

6. ContactFormTests (Feature Tests):

- **Validation of Required Fields:** Verifies that all required fields (name, email, phone, pet type, message) return validation errors when left empty.
- **Data Storage in Database:** Confirms that valid form submissions are correctly stored in the contacts table and a success message is displayed.

- **Success Message on Submission:** Ensures successful submissions redirect to the contact page with a "Thank you" success message.

7.RouteTest (Unit Tests)

- **test_home_page_route_accessible**
  Verifies that the home route is accessible for authenticated users. Ensures the route returns an HTTP status of 200 OK and renders the expected view (home). Authentication is simulated using a test user created with a factory.
- **test_about_page_accessibility**
  Ensures the about route returns a successful HTTP status (200 OK) and renders the correct view (about). Validates the accessibility and functionality of the about page.
- **test_services_page_accessibility**
  Tests the services route to confirm it returns an HTTP status of 200 OK and renders the expected view (services). Ensures the services information page is functioning as intended.
- **test_products_index_page_accessible**
  Verifies that the products.index route returns a successful HTTP status (200 OK). Ensures the products listing page is accessible and working correctly.
- **test_gallery_page_accessibility**
  Confirms the gallery route is functional, returning an HTTP status of 200 OK. Ensures the correct view (gallery) is rendered, allowing users to access the gallery content.
- **test_contact_page_accessibility**
  Validates the contact route functionality by checking it returns a successful HTTP status (200 OK) and renders the expected view (contact). Ensures users can reach the contact page for inquiries or feedback.

8.AuthenticatedUserTest (Feature Tests)

- **test_authenticated_user_can_access_product_creation_form**
  Verifies that an authenticated user can access the product creation form. Checks the products.create route and ensures it returns an HTTP status of 200 OK.
- **test_authenticated_user_can_store_a_product**
  Tests the ability of an authenticated user to store a product in the database:

  - Sends a POST request to the products.store route with valid product data (e.g., name, price, category, description).
  - Confirms the response redirects to the products index route.
  - Validates that the product is correctly stored in the database using assertDatabaseHas.

- **test_authenticated_user_can_edit_a_product**
  Confirms that an authenticated user can access the edit form for an existing product:
  - Uses the products.edit route and ensures the response status is 200 OK.
  - Validates that the product's details are ready for modification.
- **test_authenticated_user_can_update_a_product**
  Ensures that an authenticated user can successfully update an existing product's details:
  - Sends a PUT request to the products.update route with updated data (e.g., name, category, description, price).
  - Confirms the response redirects to the products index route.
  - Validates the product updates in the database using assertDatabaseHas.
- **test_authenticated_user_can_delete_a_product**
  Verifies that an authenticated user can delete an existing product:
  - Sends a DELETE request to the products.destroy route for a specific product.
  - Confirms the response redirects to the products index route.
  - Validates that the product is removed from the database using assertDatabaseMissing.