PROJECT REPORT

To fulfill the assignment for the Server Side Internet Programming course

Lecturer/Course Instructor: Williem

Compiled by:

Johana Veronica Setiawan	(001202400206)
Keira Nevrada Lay	(001202400170)
Yosmia Khaira Nisa	(001202400108)

PRESIDENT UNIVERSITY

FACULTY OF COMPUTING

CIKARANG UTARA

2025

TABLE OF CONTENTS

Github Link: https://github.com/keinvl/database_ssip_project.git

PART A

INTRODUCTION

A.1. OVERVIEW

PallPaws is a database-driven web application developed to support the digital transformation of pet shop services. This application is created using PHP and employs the Laravel framework, along with a connection to a MySQL database. It showcases the implementation of essential CRUD operations across different components, serving as a comprehensive case study for designing database-driven applications.

The system is structured to give customers an easy-to-use and effective platform for buying pet products, scheduling pet care services, and interacting with the store`s administrator. All activities and transactions performed by users are systematically stored in a well-organized MySQL database. This is associated with Laravel's Eloquent ORM, which streamlines database operations and contributes to more organized code.

PallPaws is a cutting-edge web platform that skillfully takes care of data operations.It leverages PHP to provide flexibility, organizes itself using Laravel's MVC pattern, and uses MySQL for consistent and secure data storage. This combination guarantees effective performance and simplicity in maintenance.

A.2. BUSINESS FUNCTION

The system should allows users to:

- Registration

  Allows users to create an account and access premium features such as the booking form, shopping cart, and product checkout.

- Login

  Allows users to access their account to view their shopping history and booking status.

- Forgot Password

  Allows user to recover their account their account with a new password when they losing their an old password

- Product Search

  Allows users to search for specific products they desire directly based on the name of the product.

- Filtering

  Allows users to find products based on the categories of the product such as food or clothes.

- Booking

  Allows users to choose a service, date/time, and book the selected service.

- Product Catalog

  Allows users to find all product categories with product image, product name, short description about the product and price.

- Shopping Cart

  Allows users to save products before purchasing, as well as adding and removing total products based on their desire.

- Contact Us

  Allows users to send messages relating to consultations, feedback, suggestions or complaints about products or services.

- Log-out

  Allows users to log out their account whenever they want, and revert them to the login page before they log in.

The system should allows admin to:

- Admin can access the admin page to monitor the user data
- Admin booking

  Allows admin to view and mark services that have been booked. Or delete bookings when needed.

A.3. DATA REQUIREMENTS

1. User Account Information (email, username, name, and password)
2. User Register Information (email, username, name, and password)
3. User Login Information (email, password)
4. User Booking Information (id, customer, date, time status)
5. Admin Booking Information (id, customer, date, time, confirm, delete)
6. Product Information (product image, product name, short description, price)
7. Shopping Cart Information (product image, product name, product price, total product, total price)
8. Contact Us Information (text).

A.4. BUSINESS RULES

1. The user should be able to **create an account** with the correct email address, username, and password. After filling and successfully creating an account, the user will have a unique identifier to distinguish one account from another.
2. The user should be able to **log in** with the correct email and correct password that have registered in the register page.
3. The user should be able to **access the forgotten password** page if the user loses their old password.
4. Users should be able to access the **"Home"** page after completing the registration or login page, which includes a lot of information about the website and the author information.
5. The user should be able to **access the "Pet Care" page** that has information about treatment we have available as well as booking treatments.
6. The user should be able to **access the booking form and fill out that form** to make a new reservation for our services.
7. The users should be able to **enter the dates of the services** that suit their schedule in the booking form as well as **start time and end time** of the treatment.
8. The users should be able to **choose treatment that is available** in our pages such as "Pet Grooming", "Dog Walking", "Pet Sitting", "Overnight Care", "Pet Training" or "Pet Clinic" in the booking form.
9. The user should be able to **submit booking treatments** they want to do with their pet.
10. The user should be able to **see their name on customer`s booking time tables** when they submit a booking with status "Requested".

11. The user should be able to **access "Blog" page** that have information about pets.
12. The user should be able to **access the "Shop" page** which contains catalog products with images and short descriptions of the products.
13. The user should be able to **search for specific products** they desire directly based on the name of the product.
14. The user should be able to **explore the product** based on the categories of the products.
15. The user should be able to **add several products** based on their requirement to the cart
16. The user should be able to **access the cart** when they click cart icon
17. The user should be able to **reduce and add** more products from the cart
18. The user should be able **to checkout the product** they add to cart and get success notification
19. The user should be able to **access the "Contact Us" page and write messages** such as feedback, suggestions, questions, complaints or anything the user writes.
20. The user should be able to **send the message**
21. The user should be able to **log out from their account** and **back to the login page**
22. The admin should be able to **access admin page** when they type /admin/bookings
23. The admin should be able to **monitoring the user data**
24. The admin should be able to **"Mark the Booked" from user requested**
25. The admin should be able to **delete the user requested** if necessary.

PART B

CONCEPTUAL DATA MODELLING

B.1. ERD (ENTITY RELATIONSHIP DIAGRAM)

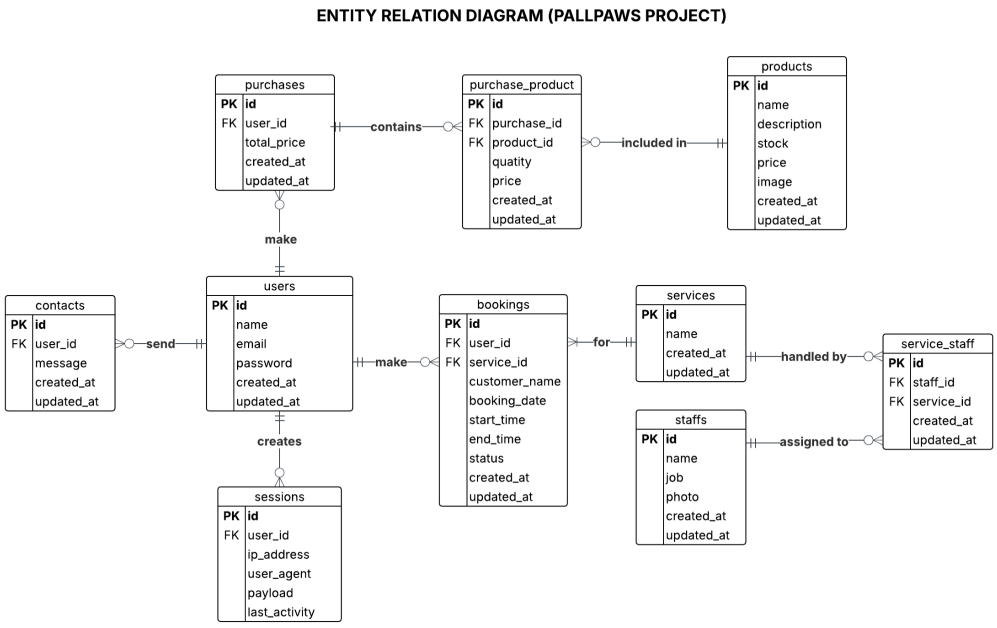

Image B.1.1: Entity Relational Diagram PallPaws Project

B.2. JUSTIFICATIONS OF ERD BASED ON BUSINESS RULES AND ASSUMPTIONS

a. BUSINESS RULES RELATED TO ENTITY USERS

BR1 : The user should be able to **create an account** with the correct email address, name, and password. After filling and successfully creating an account, the user will have a unique identifier to distinguish one account from another.

- The user creates an account with an email address and password. That justifies the existence of name, email, and password.
- The “unique identifier” justifies the existence of users_id

- The user should fill valid data when registering

BR2: The users should be able to log in with the correct email and correct password that have registered in the register page.

- Email and password attributes used on user authentication process
- When login clicked, system will match email and password with data in users table

BR3: The user should be able to access the forgot password page if the user forgot their password.

- This justifies the use of the email attribute to identify the user.
- A password reset token or workflow can be stored temporarily (not explicitly shown in ERD but typically part of system logic).
- Password field will be updated after successful password reset.

BR5: The user should be able to access the “Pet Care” page that has information about treatment we have and also book the treatment.

- The user's ability to view treatments and book them links to the bookings table.
- Each booking record is associated with a user_id establishing the relationship between users and bookings.

BR6–10: The user should be able to add treatment form, enter date, time, and select treatment.

- The user inputs booking_date, start_time, end_time, and selects a service (service_id) from a list.
- These details are stored in the bookings table, which is linked to the user through user_id.

BR11: The user should be able to submit booking treatments they want to do with their pet.

- When the user submits the form, a record is created in the bookings table with user_id, treatment details, and a status (e.g., “Requested”).

BR12: The user should be able to see their name on customer`s booking time tables when they submit a booking with status “Requested”.

- The customer_name and status fields in the bookings table allow this to be displayed.
- user_id in bookings links back to the users entity for user identification.

BR16: The user should be able to add several products based on their requirement to the cart.

- The purchases table contains user_id, product_id, and quantity, allowing users to add multiple products.
- total_price can be calculated and stored in the same table.

BR18: The user should be able to checkout the product they add to cart and get success notification.

- On checkout, the purchases table logs the finalized products with user_id.
- The system can display a success message after storing the purchase record.

BR20: The user should be able to access the “Contact Us” page and write messages.

- The contacts table allows storing a message from the user, with the user_id as a foreign key.
- Other fields like email, created_at, and updated_at track and associate the message with a user.

BR21: The user should be able to send the message.

- Justifies inserting a new record in the contacts table with user_id and message.
- created_at is used to track the submission time.

BR22: The user should be able to log out from their account and back to the login page.

- The sessions table records user sessions with the user_id, allowing session management.
- Fields like ip_address, user_agent, payload, and last_activity track session details.

BR24: The admin should be able to monitor the user data.

- The users table contains all user profile information: username, email, email_verified_at, created_at, updated_at.

- Admins can use this data to view and manage users.

BR25: The admin should be able to “Mark the Booked” from the user requested.

- Admin updates the status field in the bookings table from “Requested” to “Booked” for the selected user record.
- user_id in bookings still points to the related user.

BR26: The admin should be able to delete the user requested if necessary.

- Admin deletes the bookings record associated with a user_id as necessary.
- Deleting a booking does not remove the user from the users table — only their request.

Primary Key : user_id

b. BUSINESS RULES RELATED TO ENTITY BOOKINGS

BR7: The users should be able to enter the date they want to do the treatment.

- This requires the booking_date attribute in the bookings entity to specify when the treatment is scheduled.

BR8: The users should be able to enter the start time and end time of the treatment.

- This requires the start_time and end_time attributes in the bookings entity to indicate the timing of the treatment.

BR9: The users should be able to choose treatment that is available on our pages such as “Pet Grooming”, “Dog Walking”, “Pet Sitting”, “Overnight Care”, “Pet Training” or “Pet Clinic”.

- This requires the service_id attribute in the bookings entity to reference the specific treatment/service chosen by the user.

BR10: The user should be able to submit booking treatments they want to do with their pet.

- This requires the following attributes in the bookings entity:
    - user_id: A foreign key referencing the user making the booking.
    - customer_name: The name of the user or pet owner making the booking.
    - status: To indicate the current state of the booking (e.g., "Requested").

BR11: The user should be able to see their name on customer`s booking timetables when they submit a booking with status “Requested”.

- This relates to the status attribute in the bookings entity, which will be set to "Requested" upon submission.

BR25: The admin should be able to mark a booking as "Booked".

- The admin can update the status attribute in the bookings entity to reflect that the treatment has been confirmed.

BR26: The admin should be able to delete a booking if necessary.

- The admin can remove a booking entry using the unique id of the booking, allowing for effective management of bookings.

Primary Key : booking_id

c. BUSINESS RULES RELATED TO ENTITY SERVICES

BR9: The users should be able to **choose treatment that is available** in our pages such as “Pet Grooming”, “Dog Walking”, “Pet Sitting”, “Overnight Care”, “Pet Training” or “Pet Clinic”

- The treatment or service users can choose from are taken from the services table.
- The name field in the service table contains the list of treatments displayed in the website.

BR10: The user should be able to **submit booking treatments** they want to do with their pet.

- The relationship between the bookings and service tables is established through the service_id field in the bookings table.
- When users make a booking, they select one of the services from the services table.

BR11: The user should be able to **see their name on customer`s booking time tables** when they submit a booking with status “Requested”.

- The booking data displayed to the user will show the service name that user has selected.

BR25-26: The admin should be able to **"Mark the Booked" from user requested** and can **delete the user requested** if necessary.

- The admin manages the booking status related to a specific service from the service table.

Primary Key : service_id

d. BUSINESS RULES RELATED TO ENTITY STAFFS

BR9: The users should be able to **choose treatment that is available** in our pages such as "Pet Grooming", "Dog Walking", "Pet Sitting", "Overnight Care", "Pet Training" or "Pet Clinic"

- Although user do not select staff directly, the presence of staff with certain skills affect service availability, Admin need to ensure competent staff are assigned to the appropriate service via service_staff

BR25: The admin should be able to **"Mark the Booked" from user requested**

- After the user booking is confirmed by the admin, admin needs to assign the right staff to run the service based on available and their skills.

Primary Key : staff_id

e. BUSINESS RULES RELATED TO ENTITY SERVICE_STAFF

BR9: The users should be able to **choose treatment that is available** in our pages such as “Pet Grooming”, “Dog Walking”, “Pet Sitting”, “Overnight Care”, “Pet Training” or “Pet Clinic”.

- Admin uses service_staff to ensure each service has qualified staff assigned, so users can always book services that are actually available.

BR10: The user should be able to **submit booking treatments** they want to do with their pet.

- When booking is made, the system can check which staff are assigned for the chosen service via service_staff.

BR25: The admin should be able to **“Mark the Booked” from the user requested.**

- This assignment ensured that only qualified staff as service_staff handle the booking.

BR26: The admin should be able to delete the user requested if necessary.

- If a booking is deleted, the relationship between to service_staff ensures that staff availability is updated accordingly for future bookings.

Primary Key : service_staff_id

f. BUSINESS RULES RELATED TO ENTITY PURCHASES

BR19: The user should be able **to checkout the product** they add to cart and get success notification.

- When a user checks out, a new record is inserted into the purchases table, linking the purchase to the user and the product bought.

BR24: The admin should be able to **monitoring the user data**

- Admin can monitor all purchase transactions by joining purchases with users and products to see who bought the product, how much the product and when users checkout

Primary Key : puchases_id

g. BUSINESS RULES RELATED TO ENTITY PURCHASE_PRODUCT

BR16: The user should be able to **add several products** based on their requirement to the cart.

- The cart can contain multiple products, each represented as a row in the purchase_product table.
- Each row links a purchase to a product, with specific quantity and price.

BR19: The user should be able **to checkout the product** they add to cart and get success notification.

- When a user checks out, all items in the art (all purchase_product rows for a given purchase_id) are finalized as part of the purchase.

BR24: The admin should be able to **monitor the user data.**

- Admin can see all products bought in each purchase, including quantity and price by joining purchase_product with purchase, users, and product.

Primary Key : puchase_product_id

h. BUSINESS RULES RELATED TO ENTITY PRODUCT

BR14-15 : The user should be able to **search for specific product** directly and **explore the product** based on the type.

- The products table stores all product information, enabling users to search and explore available products

BR16 : The user should be able to **add several products** based on their requirement to the cart.

- When users add items to their cart, the system checks the products table for product details and available stock.

BR17 : The user should be able to **access the cart** when they click the cart icon.

- Each time a user updates their cart, the system references the products table to ensure stock availability and to display product details.

BR18 : The user should be able **to checkout the product** they add to cart and get success notification.

- On checkout, the system deducts the purchased quantity from the product stock in the products table.

BR24: The admin should be able to **monitoring the user data**

- Admin can monitor product sales and stock levels by joining products with purchases.

Primary Key : product_id

i. BUSINESS RULES RELATED TO ENTITY CONTACTS

BR20: The user should be able to access the "Contact Us" page and write messages.

- When a user writes a message on the Contact Us page, the system saves this message in the contact table.
- The table records the user_id, their email, their message content and the timestamp.

BR21: The user should be able to send the message.

- When the user clicks "Send" on the contact form, the message is stored in the contact table as a new record.

BR24: The admin should be able to **monitoring the user data.**

- Admin can monitor all messages sent by the users by querying the contacts table optionally joined with the users table for more user details.

Primary Key : contact_id

j. BUSINESS RULES RELATED TO ENTITY SESSIONS

BR22: The user should be able to **log out from their account** and **back to the login page**.

- When users log in, a session record is created or updated in the sessions table
- When users log out, their session can be invalidated or removed
- The system uses sessions to manage active user sessions securely

BR24: The admin should be able to **monitor the user data.**

- Admin can monitor active sessions to see who is currently logged in.

PART C

FLOW AND LOGICAL DESIGN

C.1. RELATIONS (LOGICAL DESIGN / SCHEMA CONVERSION)

Users (id, name, email, password, created_at, updated_at)

Contact (id, user_id, message, created_at, updated_at)*

User_id references users

Sessions (id, user_id, ip_address, payload, last_activity)*

User_id references users

Bookings (id, user_id, service_id*, customer_name, booking_date, start_time, end_time, status, created_at, updated_at)*

User_id references users

Service_id references services

Services (id, name, created_at, updated_at)

Service_staff (id, staff_id, service_id*, created_at, updated_at)*

Staff_id references staffs

Service_id references services

Staffs (id, name, job, photo, created_at, updated_at)

Purchases (id, user_id, total_price, created_at, updated_at)*

User_id references users

Purchase_product (id, purchase_id, product_id*, quantity, price, created_at, updated_at)*

Purchase_id references purchases

Product_id references products

Products (id, name, description, stock, price, image, created_at, updated_at)

ERD RELATION

users → bookings

- One user can create multiple bookings
- Each booking must be associated with exactly one user (mandatory 1:M).

users → purchases

- A user may perform multiple purchases throughout their usage of the platform (1:M).

purchases → purchase_product → products

- Purchases may contain many products, and each product can appear in multiple purchases.
- purchase_product serves as a bridge entity capturing quantity and price.

users → contacts

- One user can send multiple messages.

users → sessions

- Each user can have multiple session records for activity tracking and logout.

bookings → services

- Every booking is linked to a specific service (e.g., Pet Sitting, Pet Training) as per user selection.

services ↔ staffs (via service_staff)

- Many-to-many relationship to record which staff are assigned to handle which services, supporting backend management.

C.2. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO EACH BUSINESS RULES

a. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO USERS

BR1: The user should be able to **create an account** with the correct email address, name, and password.

Functional Dependencies:

- *user_id → name, email, password*
- *email → user_id, name, password (email must be unique for each user)*

BR2: The user should be able to **log in** with the correct email and password.

Functional Dependency:

- *email → password (email identifies the account and is used for authentication)*

BR3: The user should be able to **access the forgotten password page**.

Functional Dependency:

- *email → password (email is used to reset and update the password*

BR5: The user should be able to access **the Pet Care page and book treatment.**

Functional Dependency (via relation with bookings):

- *user_id → bookings*

BR6–BR10: The user should be able to **add treatment, select date/time, and treatment type**.

Functional Dependency (via relation with bookings):

- *user_id → booking_date, start_time, end_time, service_id (in the bookings table)*

BR11: The user should be able to **submit booking treatment**.

Functional Dependency:

- *user_id → status (e.g., Requested – saved in bookings)*

BR12: The user`s name should **appear in the booking schedule** upon submission.

Functional Dependency:

- *user_id → customer_name (in the bookings table)*

BR16: The user should be able to **add multiple products to the cart**.

Functional Dependencies (via purchases*):*

- *user_id → product_id, quantity*

BR18: The user should be able to **checkout the cart and get a success notificatio**n.

Functional Dependency:

- *user_id → total_price*

BR20: The user should be able to **access the Contact Us page and write messages**.

Functional Dependency (via contacts table):

- *user_id → email, message*

BR21: The user should be able to **send the message**.

Functional Dependency:

- *user_id → created_at (timestamp of message submission)*

BR22: The user should be able to **log out and return to the login page.**

Functional Dependency (via sessions):

- *user_id → ip_address, user_agent, payload, last_activity*

BR24: The admin should be able to **monitor user data.**

Functional Dependency:

- *user_id → name, email, email_verified_at, created_at, updated_at*

BR25: The admin should be able t**o "Mark the Booked" from user requested**

Functional Dependency:

- *user_id → status (status field in bookings)*

BR26: The admin should be able to delete the user requested if necessary

Functional Dependency:

- *user_id → bookings (bookings can be deleted by admin)*

b. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO BOOKINGS

BR7: The users should be able to **enter the date** they want to do the treatment.

Functional Dependency:

- *booking_id → booking_date*

BR8: The users should be able to **enter the start time and end time of the treatment.**

Functional Dependency:

- *booking_id → start_time, end_time*

BR9: The users should be able to **choose treatment that is available** on our pages such as "Pet Grooming", "Dog Walking", "Pet Sitting", "Overnight Care", "Pet Training" or "Pet Clinic".

Functional Dependency:

- *booking_id → service_id*

BR10: The user should be able to **submit booking treatments** they want to do with their pet.

Functional Dependency:

- *booking_id → user_id, customer_name, status*

BR11: The user should be able to **see their name on customer`s booking timetables** when they submit a booking with status "Requested".

Functional Dependency:

- *booking_id → status*

BR25: The admin should be able t**o "Mark the Booked" from user requested**

Functional Dependency:

- *booking_id → status*

BR26: The admin should be able to **delete a booking** if necessary.

Functional Dependency:

- *booking_id → all attributes in bookings (used for identifying and removing the record)*

c. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO SERVICES

BR9: The users should be able to **choose treatment that is available** in our pages such as “Pet Grooming”, “Dog Walking”, “Pet Sitting”, “Overnight Care”, “Pet Training” or “Pet Clinic”.

Functional Dependency:

- *service_id → name*

BR10: The user should be able t**o submit booking treatments** they want to do with their pet.

Functional Dependency:

- *service_id → name (used in bookings to show selected treatment)*

BR11: The user should be able to **see their name on customer`s booking time tables** when they submit a booking with status “Requested”.

Functional Dependency:

- *service_id → name (displayed alongside booking info)*

BR25–26: The admin should be able to **“Mark the Booked” from user requests** and can **delete the user** requested if necessary.

Functional Dependency:

- *service_id → name (used to manage and identify treatment type for each booking)*

d. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO STAFFS

BR9: The users should be able to **choose treatment that is available** in our pages such as “Pet Grooming”, “Dog Walking”, “Pet Sitting”, “Overnight Care”, “Pet Training” or “Pet Clinic”.

Functional Dependency:

- *staff_id → name, position, photo (used by admin to ensure staff have correct skills via service_staff relationship)*

BR25: The admin should be able to **“Mark the Booked” from the user requested.**

F**unctional Dependency:**

- *staff_id → name, position (admin selects appropriate staff based on availability and skills to handle confirmed bookings)*

e. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO SERVICE_STAFF

BR9: The users should be able to **choose treatment that is available** in our pages such as “Pet Grooming”, “Dog Walking”, “Pet Sitting”, “Overnight Care”, “Pet Training” or “Pet Clinic”.

Functional Dependency:

- *service_staff_id → service_id, staff_id*

BR10: The user should be able t**o submit booking treatments** they want to do with their pet.

Functional Dependency:

- *service_staff_id → service_id, staff_id (system checks this relation to validate staff assignment for booked service)*

BR25: The admin should be able to **"Mark the Booked" from user requested.**

Functional Dependency:

- *service_staff_id → staff_id (ensures only appropriate staff handle the confirmed booking)*

BR26: The admin should be able to **delete the user requested** if necessary.

Functional Dependency:

- *service_staff_id → staff_id, service_id*

f. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO PURCHASES

BR19: The user should be able to **checkout the product they add to cart and get success notification.**

Functional Dependency:

- *purchases_id → user_id, product_id, quantity, total_price, created_at*

BR24: The admin should be able to **monitor the user data.**

Functional Dependency:

- *purchases_id → user_id, product_id, total_price, created_at (enables tracking of who bought what and when)*

g. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO PURCHASE_PRODUCT

BR16: The user should be able to **add several products** based on their requirement to the cart.

Functional Dependency:

- *purchase_product_id → purchase_id, product_id, quantity, price*

BR19: The user should be able to **checkout the product they add to cart and get success notification.**

Functional Dependency:

- *purchase_product_id → purchase_id, product_id, quantity, price*

BR24: The admin should be able to **monitor the user data.**

Functional Dependency:

- *purchase_product_id → product_id, quantity, price*

h. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO PRODUCT

BR14–15: The user should be able to **search for specific products directly and explore the product based on the type.**

Functional Dependency:

- *product_id → name, description, image*

BR16: The user should be able to **add several products** based on their requirement to the cart.

Functional Dependency:

- *product_id → stock, name, image*

BR17: The user should be able to **access the cart when they click cart icon.**

Functional Dependency:

- *product_id → stock, name, image*

BR18: The user should be able to **checkout the product they add to cart and get success notification.**

Functional Dependency:

- *product_id → stock*

BR24: The admin should be able to **monitor the user data.**

Functional Dependency:

- *purchase_product_id → product_id, quantity, total_price (used for viewing product details in a purchase*)

i. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO CONTACTS

BR20: The user should be able to access the "Contact Us" page and write messages

Functional Dependency:

- *contact_id → user_id, email, message, created_at (used to store contact message input from the user)*

BR21: The user should be able to send the message

Functional Dependency::

- *contact_id → user_id, email, message (used to store submitted contact form as a new record*)

BR24: The admin should be able to monitor the user data

Functional Dependency:

- *contact_id → user_id, email, message (used for viewing user messages, optionally joined with users table for more context*)

j. LIST OF FUNCTIONAL DEPENDENCIES RELATED TO SESSIONS

BR22: The user should be able to **log out from their account and back to the login page**

Functional Dependency:

- *session_id → user_id, ip_address, user_agent, payload, last_activity*

BR24: The admin should be able to **monitor the user data**

Functional Dependency:

- *session_id → user_id, last_activity (used to monitor currently active user sessions*)

C.3. SYSTEM FLOW

This section explains the logical sequence of operations for each feature implemented within the PallPaws web application.

a. User Login Page

Upon their initial login, users will be directed to the login page to verify their details. The login system verifies whether the email and password provided by users correspond with the information saved in the database. Once the user login successfully, a session is initiated to ensure the user's identity is maintained.

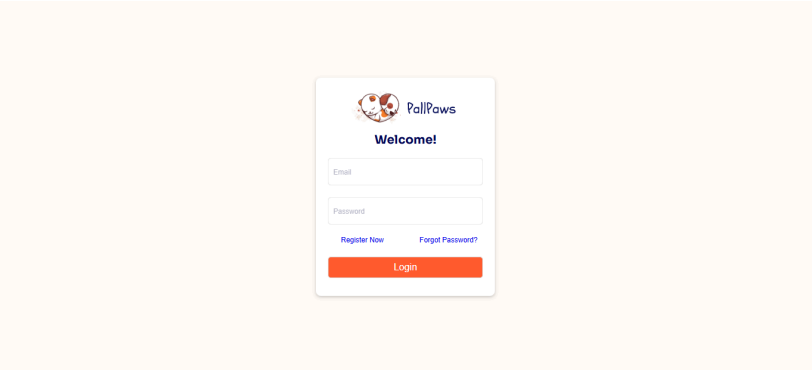

Image C.3.1: Log-in Page

b. User Registration Page

For users who do not have an account, the system provides a “Register Now” feature. Users are asked to complete a registration form by entering essential information such as a unique name, email address, and a secure password with confirmation. If the gmail account is already in use, the system will notify the user that the email is taken.When the registration is completed, user will proceed to the login page.

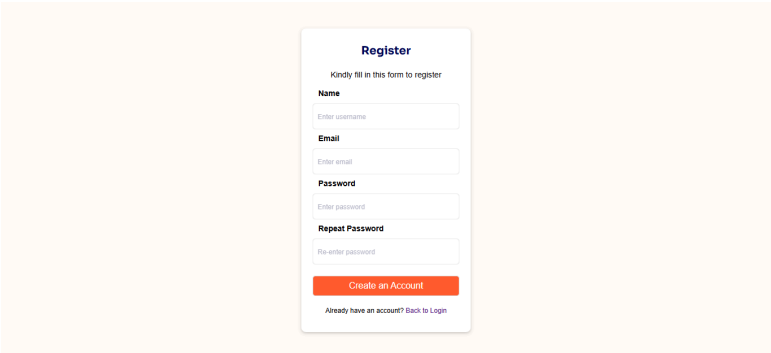

Image C.3.2: Register Page

c. Forgot Password Page

In case the user cannot recall their password. There is a “Forgot Password” feature provided by PallPaws. Users can update their password by providing their email address and entering the new password twice for confirmation. The system validates email existence, then updates the corresponding record with a new hashed password.

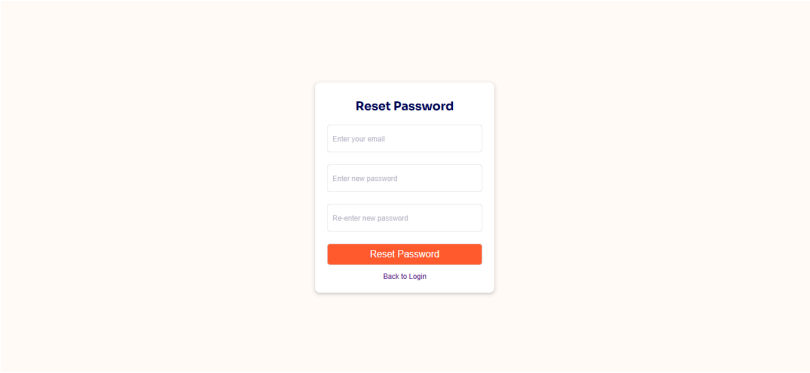

Image C.3.3: Forgot Password Page

d. Home Page

After the user successfully logs into the account, the user will access the home page. It's a greeting page that provides an overview of a website's content.

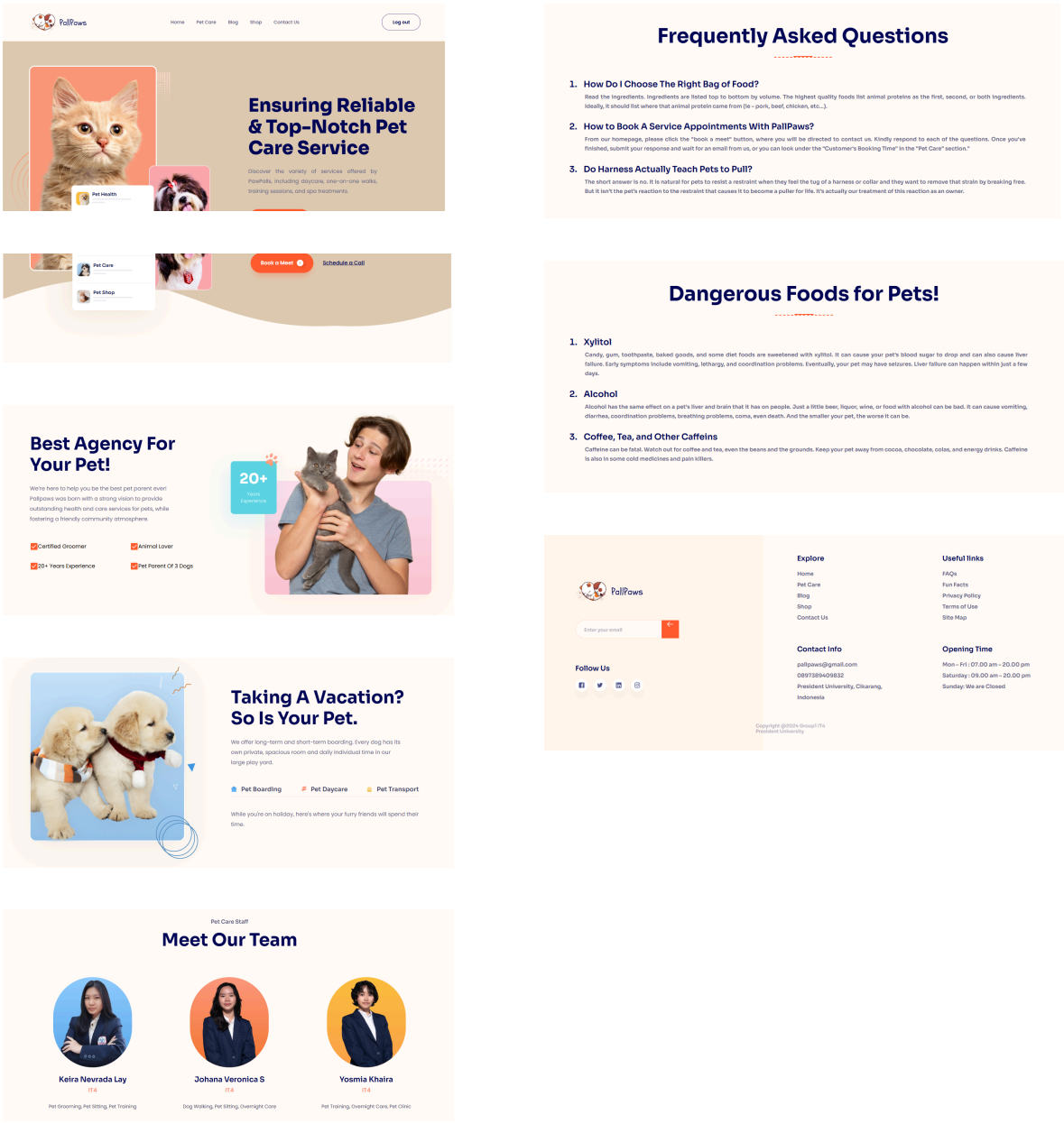

Image C.3.4: Home Page

e. Pet Care Page

PallPaws offers a booking feature that enables users to reserve various pet services, including grooming, dog walking, pet sitting, overnight care, pet training, and veterinary appointments directly through web application. Authenticated users can

access a booking form, where they are required to select the service they desire, pick a date, and determine schedule time. Once the form is submitted, the system records the booking details into the booking table within the relational database, linking it to the corresponding user and service entries via foreign keys. This procedure streamlines the scheduling of services, ensuring that all reservations are well-organized, easily accessible, and manageable.

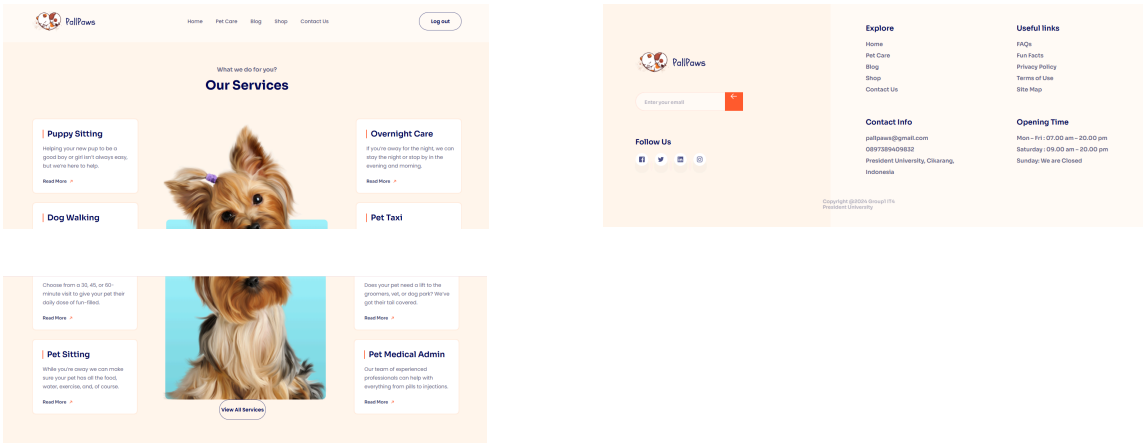

Image C.3.5: Pet Care Page

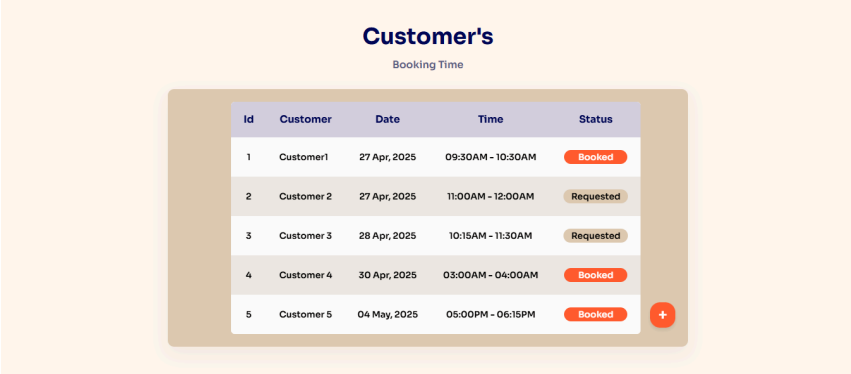

Image C.3.6: Booking Form

f. Admin View Page

PallPaws has implemented a system for managing bookings, allowing administrators to track and control all user service appointments. The function allows administrators to access booking records directly from the database. Administrators have access to information including the service type, customer data, appointment date and time, as well as to change the status of the reservation and delete the appointments.

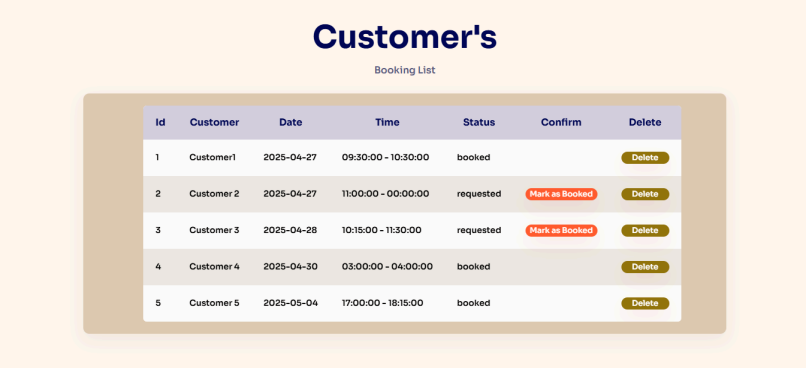

Image C.3.7: Admin Booking Form

g. Shop Page

The shop feature in PallPaws functions as the core of its e-commerce component, allowing users to browse, search, and purchase a variety of pet-related products. Product information such as name, description, price, and image is dynamically retrieved from the product table in the database and shown in an organized way.

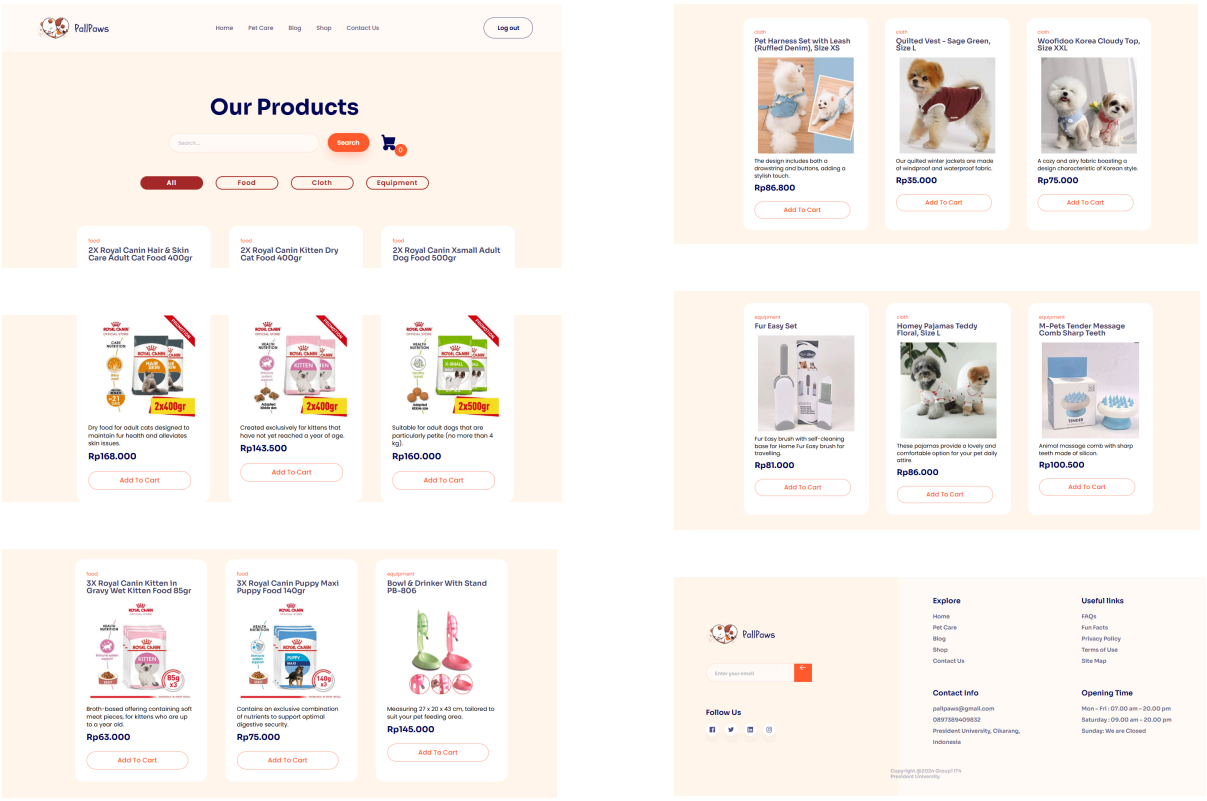

Image C.3.8: Shop Page

h. Cart

When a customer adds products to their shopping cart and proceeds to checkout, the transaction information is recorded in a purchases table. This assists in monitoring both the inventory levels and the orders placed by customers.

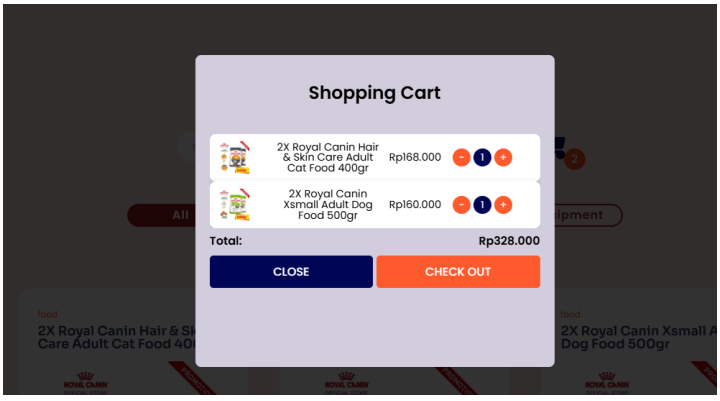

Image C.3.9: Shopping Cart

i. Contact Us Page

Customers can also get in touch with the PallPaws staff. Users have the option to complete a form to inquire, provide feedback, or communicate any issues they may have. Essential information such as their name, email address, gender, type of pets they own, and message must be included in the form. The information sent undergoes a verification process for accuracy before being stored in the contact_us table within the database.

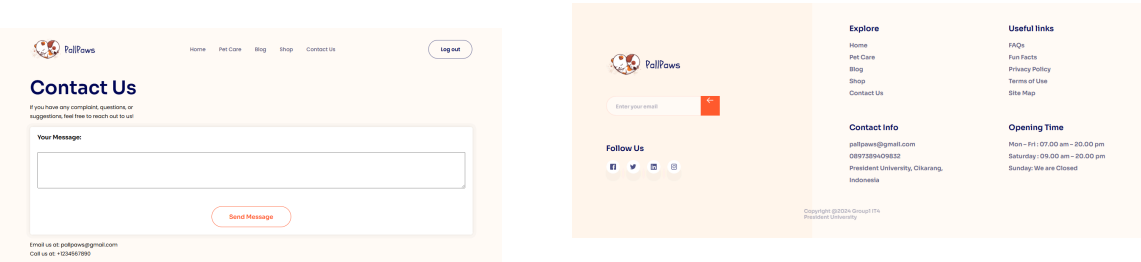

Image C.3.10: Contact Us Page

To exit the website, the user can simply click the logout button located in the top right corner. Once a user logs out, the system concludes their session and deletes all authentication tokens information, ensuring they are entirely disconnected from the application.

Image C.3.11: Log-out

C.4. CRUD OPERATION

The implementation of CRUD (Create, Read, Update, Delete) actions is essential in the development of PallPaws. They facilitate interaction between the user interface and

the database. One of the essential aspects of the system that extensively employs CRUD operations is the Booking functionality. This function allows users to schedule various pet services such as grooming, walking, sitting, and veterinary appointments. This section will detail the usage of every CRUD operation in the booking system, along with functional code examples.

a. Create

The create operation allows authenticated users to schedule a new booking by submitting a form on the “Pet Care” page. This form includes fields for entering the customer's name, the service requested, the booking date, as well as the start and end times.

Once you submit the form, the store() function in the BookingController is executed. Laravel offers a capability that ensures all essential details are filled out correctly while confirming the availability of the selected service in the database.

```
public function store(Request $request)
    {
        $request->validate([
            'customer_name' => 'required',
            'booking_date' => 'required|date',
            'start_time' => 'required',
            'end_time' => 'required',
            'service_id' => 'required|exists:services,id',
        ]);

        Booking::create([
            'user_id' => Auth::id(),
            'customer_name' => $request->customer_name,
            'booking_date' => $request->booking_date,
            'start_time' => $request->start_time,
            'end_time' => $request->end_time,
            'service_id' => $request->service_id,
```

```
        'status' => 'requested',
    ]);

    return redirect()->back()->with('success', 'Booking created successfully!');
}
```

The system employs Eloquent ORM to create a new entry in the bookings table, automatically associating it with the user who is currently logged in (Auth::id()).New bookings will be classified as "requested" until an administrator addresses them.

b. Read

The read feature in Laravel allows the system to retrieve and display saved booking records in two primary methods: through the user's interface and the admin dashboard.

- User-side reading

  Users can view their historical and upcoming bookings, along with a selection of available services. This is achieved using the show() method:

```
public function show()
    {
        $services = Service::all();
        $bookings = Booking::with('service')->where('user_id', Auth::id())->get();

        return view('pages.booking', compact('services', 'bookings'));
    }
```

In this instance, Laravel's Eloquent ORM retrieves all services and narrows down the bookings to display only those associated with the currently logged-in user. Using the with ('service') method fetches related service information in one

comprehensive query. This minimizes the burden on the database and accelerates processing speed.

- Admin-side reading

Administrators have the ability to view all user reservations through the index() function.This attribute is essential for organizing the appointments arranged using the system:

```
public function index()
    {
        $bookings = Booking::with(['user', 'service'])->get();
        return view('admin.bookings', compact('bookings'));
    }
```

By consolidating all booking data with corresponding user and service specifics, this method offers the administrator a thorough understanding of the situation.

c. Update

The update function enables administrators to modify a booking's status from "requested" to "booked." This phase is crucial for managing a service request and confirming the approval of the appointment.

```
public function updateStatus($id)
    {
        $booking = Booking::findOrFail($id);
        if ($booking->status === 'requested') {
            $booking->status = 'booked';
            $booking->save();
        }
        return redirect()->back();
    }
```

The approach initially verifies the presence of the booking by utilizing findOrFail(). An invalid ID will result in a 404 error being shown. If the status indicates "requested," it is modified to "booked" and subsequently stored. This guarantees that only pending bookings are confirmed, helping to avoid any potential mistakes.

d. Delete

The destroy() function allows administrators to completely erase a booking from the database. This method plays a vital role in rescinding appointments that are redundant or were incorrectly booked.

```
public function destroy($id)
    {
        $booking = Booking::findOrFail($id);
        $booking->delete();
        return redirect()->back();
    }
```

This procedure effectively retrieves a booking based on its ID and eliminates it using Laravel's Eloquent ORM. It is irreversible and will permanently remove the associated data from the bookings table.

PART D

CONCLUSION

The PallPaws initiative demonstrates the effective integration of PHP, the Laravel framework, and MySQL in building an engaging website that interacts with a database. It encompasses fundamental capabilities for data management tasks such as creating, reading, updating, and deleting information, along with user interaction features and a structured layout to ensure a reliable system for running a pet shop. This feature enables users to conveniently view products, book services, and interact with the admin, simplifying the experience for everyone.

Working on this project has given us important hands-on experience in creating real websites. We have strengthened our expertise in backend processes, including the development of databases and the management of pathways, as well as linking user interfaces. The experience of the project allowed us to refine our problem-solving, error-correction, and teamwork skills, all of which are vital for our professional growth ahead.

To further showcase the project, the full source code is publicly available on GitHub: https://github.com/keinvl/database_ssip_project.git. Publishing it on GitHub reflects our commitment to clean code, version control, and professional development practices.