CPT202
Final Report for Software Engineering Group Project

2022/2023 Semester 2
Project B

May 14, 2023

Group number: 9
Student Name:Jing Yin
Student ID:2034355
Project URL:

USER URLS
http://114.55.42.209:8080/[1]
http://114.55.42.209:8080/home[1]
http://114.55.42.209:8080/mainpage[1]
http://114.55.42.209:8080/account
http://114.55.42.209:8080/appointment
http://114.55.42.209:8080/agreement
http://114.55.42.209:8080/history
http://114.55.42.209:8080/groomer-schedules

MANAGER URLS
http://114.55.42.209:8080/management_system
http://114.55.42.209:8080/manager/master-file
http://114.55.42.209:8080/manager/statistical-reports
http://114.55.42.209:8080/manager/managerorders
http://114.55.42.209:8080/manager/SellingStrategy
http://114.55.42.209:8080/manager/shop-appearance/find-all
ACCESS TO ALL
http://114.55.42.209:8080/login
http://114.55.42.209:8080/logout
http://114.55.42.209:8080/help
http://114.55.42.209:8080/sign-up

Default accounts	Username	Password
Manager	realshopmanager	9js91.hn7tf-lk1mu
User (add more by signing-up)	user	password

.jar file **location**: `/root/cpt202`
The program runs in the *Tmux* window `cpt202`

[1] `/`, `/home`, and `/mainpage` all refer to the main page for users; We do this because different users may have different habits for typing home page URLs.

Contents

1 Introduction

This report offers an extensive analysis of the development and implementation of a pet grooming appointment website. Our team engaged in a collaborative approach to understand the specifications, define epics, and identify features. Following the confirmation of Product Backlog Items (PBIs), tasks were allocated to individual team members for the design and implementation of their respective features. The Scrum framework was employed throughout the project to ensure efficient project management[1].

The final solution is an intuitive and user-friendly pet grooming appointment website that streamlines the booking process for customers while simplifying appointment management and the operation process for managers while managing the appointment system. From my personal perspective, the solution's strengths lie in its seamless integration, responsiveness, and ease of use, which enhance the overall user and manager experience. However, potential weaknesses may include limited customization options and scalability constraints as the projects grows.

The report is organized into four main sections: (1) Software Development Process, delving into the methodology and workflow adopted for this project; (2) Software Design, outlining the architectural and design aspects of the website; (3) Change Management, detailing strategies for addressing modifications and improvements; and (4) Legal, Social, Ethical, and Professional Issues, discussing various aspects of compliance and responsibility.

The content covered in this report aims to provide a well-structured and insightful account of the project, highlighting key aspects of its development and shedding light on the decisions and processes that shaped the outcome. Additionally, it offers an evaluation of the system's strengths and weaknesses, reflecting on the final solution and its potential for improvement.

2 Software Development Process

Scrum is an agile framework for software development that emphasizes iterative and incremental development, team collaboration, and continuous improvement[2]. Scrum facilitates the software development process in various aspects of this project, as discussed below:

Iterative and incremental development Scrum follows iterative and incremental development approach by dividing the project development process into manageable iterations called sprints, which typically last one to four weeks[3]. During each sprint, the team works on a subset of features to deliver a potentially shippable product increment. This method enables the team to receive early and frequent feedback, allowing them to adjust and refine the product backlog as needed.

Team Collaboration Scrum values team collaboration among all members, including the product owner, development team, and Scrum Lead[3]. The team conducts a daily 15-minute standup meeting to track progress and identify any issues or obstacles that require attention. This practice promotes transparency and enables the team to quickly identify and resolve any problems. Additionally, we communicate with the Scrum Lead through office hours and post-session questions to clarify project needs further.

Continuous Improvement Scrum fosters continuous improvement through regular sprint reviews, which are conducted at the end of each sprint and last for an hour[3]. During the sprint review, the team reflects on what went well and identifies areas for improvement in the upcoming sprint. This practice promotes a culture of continuous improvement and enables the team to refine their processes over time.

3 Software Design

3.1 PBIs related to appointment

3.1.1 PBI # CS1101

PBI#	CS1101
Title	Implement a form that allows customers to enter their pet's details such as name, breed, age, description, and weight
Story	As a customer, I want to be able to easily provide all necessary information about my pet when booking an appointment with a pet grooming service so that I can make appointment efficiently.
Description	The goal of this PBI is to implement a form that allows customers to enter their pet's details when booking an appointment with a pet grooming service. The form should be user-friendly and easy to use, and should ensure that all necessary information about the pet is collected in a clear and concise manner. The form should include fields for the pet's name, breed, age, description, and weight, as well as any other relevant information that the grooming service may require.

Design

1. Add a new class called “Pet” to the project.
2. In the “Appointment” class, add a one-to-one relationship with the ”Pet” class to associate each appointment with a pet.
3. Create a new form in the front-end of the application that allows customers to enter their pet's details, such as name, breed, age, description, and weight.
4. Add validation to the form to ensure that all necessary fields are filled out correctly.

Implementation

1. Create the “Pet” class with attributes for name, breed, age, description, size, and a reference to the associated user (userId).
2. In the “Appointment” class, add a one-to-one relationship with the “Pet” class by adding a “pet” attribute with the ”@OneToOne” annotation and mapping it to the ”Pet” class.
3. In the front-end of the application, create a new form with input fields for each attribute of the ”Pet” class.
4. Use JavaScript or another front-end technology to validate the form and ensure that all necessary fields are filled out correctly.
5. Implement the “PetService” class, which includes methods such as “addPet”, “deletePet”, “updatePet”, “findPetByPetName”, and “findAllPet”.
6. Implement the “PetRepo” interface and its JpaRepository extension, including the “findByPetName” method.
7. Test the form thoroughly to ensure that it is functioning as expected and is free of bugs or errors.
8. Optimize the form for performance and accessibility. For example, consider using client-side validation to reduce server load and make the form more responsive.

Justification

1. Modularity: By creating separate classes for Pet, Appointment, PetService, and PetRepo, the system is organized into modular components. This modularity makes the system easier to understand, maintain, and scale in the future.
2. Encapsulation: The Pet and Appointment classes encapsulate the relevant data for pets and appointments, respectively. This encapsulation helps maintain data integrity and simplifies data management by grouping related information together.
3. Usability: The front-end form for entering pet details is designed to be user-friendly, ensuring that customers can easily input their pet’s information. This focus on usability contributes to a positive user experience.
4. Robustness: Adding validation to the form ensures that all necessary fields are filled out correctly, which helps maintain data integrity and reduce the likelihood of errors. This robustness contributes to a more reliable and efficient system.
5. Reusability: By implementing the PetService class with essential CRUD operations, the system becomes more versatile and reusable. The methods in the PetService class can be easily reused for different functionalities within the application.
6. Scalability: The use of JpaRepository for the PetRepo interface allows the system to take advantage of the built-in database management functionalities provided by the Spring Data JPA framework. This choice makes it easier to scale the system as needed in the future.
7. Maintainability: Thorough testing of the form and its components ensures that the system is free of bugs and errors. This attention to maintainability helps reduce the likelihood of issues arising during usage and makes the system more reliable.
8. Performance: Optimizing the form for performance and accessibility by using client-side validation improves the overall user experience. This optimization helps create a more responsive and efficient system.

Visualizing System Design: UML Diagrams These UML diagrams illustrates the following:

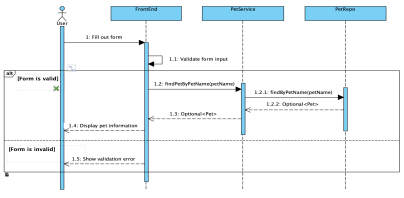

Figure 1: Pet Search Interaction Process

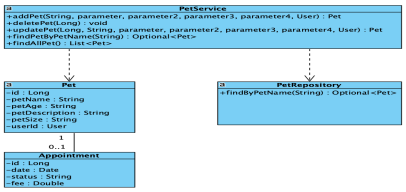

Figure 2: Pet Management System Structure

1. Sequence Diagram: This diagram illustrates the interactions between the user, front-end form, PetService, and PetRepo when searching for a pet by its name. The user fills out a form, and the front-end validates the input. If the form is valid, the front-end calls the PetService's findPetByPetName method, which then calls the PetRepo's findByPetName method. Finally, the result (Optional$\langle Pet \rangle$) is returned to the front-end and displayed to the user. If the form is invalid, a validation error is shown to the user.
2. Class Diagram:The relationships between classes are also shown in the class diagram. The Pet class has a one-to-one association with the Appointment class. The PetService class has a dependency on both the Pet and PetRepo classes.

3.1.2 PBI # CS1102

PBI#	CS1102
Title	Select appointment time
Story	As a customer, I want to be able to select an appointment time when booking a pet grooming service, so that I can schedule the appointment at a convenient time.
Description	The goal of this PBI is to enable customers to select an appointment time when booking an appointment with the pet grooming service. The system should allow the customer to view available time slots for the selected date and select a time slot that suits their schedule.

Design

1. Retrieve the available time slots for the desired employee (groomer) by calling getAppointmentsByEmployee.
2. Filter the available time slots based on the user's preferences (e.g., date, time).
3. Display the filtered time slots to the user.
4. Allow the user to choose a preferred time slot and create an Appointment instance with the selected time and associated user and employee (groomer) information.
5. Save the appointment using the saveAppointment method.

Implementation

1. The AppointmentRepo interface extends JpaRepository and provides a series of query methods for working with appointment data. It includes methods like findByEmployeeId, findByUser, and several other queries that aggregate sales data based on a given date range.
2. The AppointmentServices interface defines several methods to interact with the appointment data, including getAllAppointments, getAppointmentById, saveAppointment, deleteAppointmentById, getAppointmentsByEmployee, and getAppointmentsByUser.
3. The AppointmentServicesImpl class implements the AppointmentServices interface and uses the AppointmentRepo to perform the required operations. This class is annotated with @Service, making it a Spring Service that can be injected as a dependency in other components.
4. The UserControlle create an endpoint in a controller class that accepts user input for appointment time and other required details. This endpoint would call the saveAppointment method of the AppointmentServices interface to store the appointment in the database.

Justification

1. Separation of concerns: The different components in the design and implementation are responsible for different aspects of the system, such as handling HTTP requests, interacting with the database, and managing business logic. This ensures that each component has a clear and well-defined responsibility, which makes the system easier to understand and maintain.
2. Modularity: The use of interfaces and classes to encapsulate different aspects of the system makes it modular, which means that components can be developed and tested independently. This allows for greater flexibility and makes it easier to scale the system as it grows.
3. Abstraction: The use of interfaces and annotations, such as @Service, abstract away implementation details and allow for more generic and flexible components. This allows the system to be more easily extensible and maintainable.

4. Object-oriented design: The use of classes and objects to represent different aspects of the system allows for a more intuitive and flexible design. This allows developers to more easily reason about the system and modify it as needed.
5. Spring Framework: The use of the Spring Framework provides a number of benefits, such as dependency injection and inversion of control. This allows for greater flexibility and testability of the system, as well as easier integration with other Spring-based components.

Visualizing System Design: UML Diagram This UML sequence diagram3 helps to illustrate the flow of events in the process of selecting an appointment time. It shows the steps involved in retrieving available time slots, filtering them based on user preferences, allowing the user to select a preferred time slot, creating an Appointment instance with the necessary information, and saving the appointment to the database.

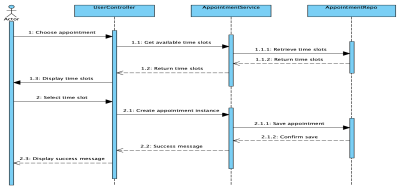

Figure 3: Appointment Time Selection Sequence Diagram

3.1.3 PBI # CS1103

PBI#	CS1103
Title	Provide a drop-down list of available services for customers to choose from
Story	As a customer, I want to have a drop-down list of available services, So that I can easily choose the service I want to book an appointment for.
Description	The purpose of this PBI is to enhance the user experience when booking an appointment by providing a drop down list of available services. This drop down list will allow customers to easily select the id corresponding to the service they want to book.

Design

1. Define a Service entity that represents the available services.
2. Create a ServiceRepo interface that extends JpaRepository to handle the CRUD (Create, Read, Update, Delete) operations related to the Service entity.
3. Create a ServiceController class that handles the HTTP requests related to the Service entity.
4. Define a method in the ServiceController class that returns a list of all available services.
5. Create a HTML form that contains a drop-down list and submits the selected value to the server.
6. Populate the drop-down list with the available services by retrieving them from the ServiceRepo and passing them to the HTML form using a Model object.

Implementation

1. Create a Service class representing a service offered by the groomers. Include properties such as id, name, price, type and description.
2. Create a ServiceRepository extending JpaRepository to manage the Service entities.
3. Create a ServiceService interface with methods such as findAllServices,addService and findServicesByType to specifies the essential methods for working with services. When implementing this interface, a developer must provide concrete implementations for all these methods, ensuring the application's service management functionality is complete and consistent.
4. Implement the ServiceService interface with a ServiceServiceImpl class. Use the ServiceRepository to perform the required operations. Additionally, The @Autowired annotation is used to automatically inject an instance of ServiceRepo into the serviceRepo variable. This enables the class to interact with the database and manage the Service entities as specified by the implemented ServiceServices interface.
5. Create a new ServiceController class with an endpoint that returns a list of all available services. The frontend will use the returned list to populate the drop-down menu.
6. On the frontend, create a drop-down list using the data received from the /services endpoint. Display the service name and price in the drop-down list. Ensure that the customer cannot book an appointment without selecting a service.
7. Modify the CreateAppointmentRequest class to include a serviceId property. This property will store the selected service's ID when the customer books an appointment.
8. Finally, update the UserController class (or create a new one if it doesn't exist) to handle the appointment creation request, including the selected service ID. Use the modified CreateAppointmentRequest class to capture the user input, including the selected service.

Justification

1. Modularity: By creating separate classes for Pet, Appointment, PetService, and PetRepo, the system is organized into modular components. This modularity makes the system easier to understand, maintain, and scale in the future.
2. Encapsulation: The Pet and Appointment classes encapsulate the relevant data for pets and appointments, respectively. This encapsulation helps maintain data integrity and simplifies data management by grouping related information together.
3. Usability: The front-end form for entering pet details is designed to be user-friendly, ensuring that customers can easily input their pet's information. This focus on usability contributes to a positive user experience.
4. Robustness: Adding validation to the form ensures that all necessary fields are filled out correctly, maintaining data integrity and reducing the likelihood of errors. This robustness contributes to a more reliable and efficient system.
5. Reusability: By implementing the PetService class with essential CRUD operations, the system becomes more versatile and reusable. The methods in the PetService class can be easily reused for different functionalities within the application, promoting code reuse and reducing duplication.
6. Scalability: The use of JpaRepository for the PetRepo interface allows the system to take advantage of the built-in database management functionalities provided by the Spring Data JPA framework. This choice makes it easier to scale the system as needed in the future.
7. Maintainability: Thorough testing of the form and its components ensures that the system is free of bugs and errors. This attention to maintainability helps reduce the likelihood of issues arising during usage and makes the system more reliable.
8. Performance: Optimizing the form for performance and accessibility by using client-side validation improves the overall user experience. This optimization helps create a more responsive and efficient system.

Visualizing System Design: UML Diagram

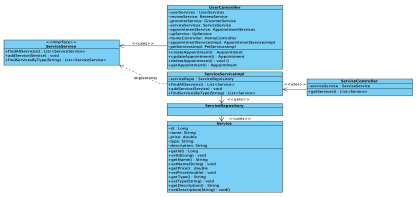

Figure 4: Service Management System

3.1.4 PBI # CS1104

PBI#	CS1104
Title	Display a list of groomers with their employee numbers and ranks for customers to choose from
Story	As a customer, I want to be able to view a list of groomers along with their employee numbers and ranks so that I can choose the one that suits my needs.
Description	The goal of this PBI is to develop a feature that allows customers to view a list of available groomers along with their respective employee numbers and grades, in order to help customers make an informed decision when choosing an groomer for their service.

Design To design a system that displays a list of groomers with their employee numbers and grades for customers to choose from, we'll use a combination of frontend and backend components.

1. Frontend:
   - HTML/CSS: Design a user-friendly page layout with clear sections and headings for displaying the groomers list.
   - JavaScript: If needed, use JavaScript to make the page more interactive, e.g., filtering or sorting the list of groomers based on user input.
2. Backend:
   - Groomer entity: Create a Groomer entity class that represents a groomer in the system. It should have fields like id, employeeId, name, rank, and a list of schedules.
   - GroomerRepository: Create an interface that extends JpaRepository and provides methods for finding groomers by rank and employeeId.
   - GroomerService: Create a service layer that interacts with the GroomerRepository to query the database and handle the business logic related to groomers.

- GroomerController: Create a RESTful API controller for handling groomer-related requests. It should provide endpoints for finding all groomers, finding groomers by employeeId or rank, adding new groomers, deleting groomers by employeeId, and editing groomers' employeeId, name, or rank.

3. Database:

   - Design a database schema with a table for groomers, including columns for id, employeeId, name, rank, and a foreign key for the list of schedules.

4. Workflow:

   - When a customer visits the page to view the list of groomers, the frontend sends a request to the GroomerController.
   - The GroomerController calls the appropriate method in the GroomerService to retrieve the required data.
   - The GroomerService interacts with the GroomerRepository to query the database for the list of groomers.
   - The GroomerRepository returns the list of groomers, which is passed back to the GroomerService.
   - The GroomerService processes the data, if necessary, and sends it back to the GroomerController.
   - The GroomerController sends the groomer data to the frontend.
   - The frontend renders the groomer data on the page using the Thymeleaf template, displaying the list of groomers with their employee numbers and grades for customers to choose from.

Implementation The components below work together to display the list of groomers with their employee numbers and grades for customers to choose from. The GroomerController handles the HTTP requests, while the GroomerRepository queries the database for the required information. The data is then passed to the Thymeleaf template, which renders the groomers in a user-friendly format.

1. The HTML template uses Thymeleaf to iterate through the groomerList and display groomers based on their rank. Each block of code displays groomers for a specific rank (1-5), showing the groomer's name and employeeId.
2. The Groomer class is an entity representing a groomer in the system. It has fields for id, employeeId, name, rank, and a list of schedules. The class includes constructors, getters, and setters for each field, as well as methods for adding, editing, and deleting groomers.
3. The GroomerController is a RESTful API controller for handling groomer-related requests. It provides endpoints for finding all groomers, finding groomers by employeeId or rank, adding new groomers, deleting groomers by employeeId, and editing groomers' employeeId, name, or rank.
4. The GroomerRepository interface extends JpaRepository and provides methods for finding groomers by rank and employeeId.

Justification

1. Modularity: By breaking down the system into distinct classes - Groomer, GroomerRepository, GroomerService, and GroomerController, we've increased modularity. This allows for a more organized and understandable system that is easier to maintain, modify, and scale in the future.
2. Encapsulation: In the Groomer class, all related data is encapsulated, maintaining data integrity and grouping related information together. This leads to easier data management and supports the principles of Object-Oriented Programming (OOP).
3. Usability: The front-end interface is designed to display a list of groomers with their employee numbers and grades in a clear and user-friendly format, facilitating an easy and efficient selection process for customers. This focus on usability ensures an improved user experience.
4. Robustness: Through form validation on the front-end and data validation on the backend, the system ensures robustness. This mitigates the risk of incorrect data being entered into the system, and reduces the likelihood of errors, contributing to a reliable system.
5. Reusability: By designing the GroomerService class with essential CRUD operations, the system becomes more versatile and reusable. The methods in this class can be reused across different functionalities within the application, promoting code reuse and reducing code duplication.
6. Scalability: The GroomerRepository interface extends JpaRepository, which provides built-in methods for database operations, allowing the system to easily scale up as necessary. It also supports pagination and sorting, which can be crucial for performance as the data size grows.
7. Maintainability: The system is designed to be maintainable, with each component having a clear responsibility. In addition, the system can be easily tested, ensuring a high level of quality and reducing the likelihood of bugs or issues arising during usage.

Visualizing System Design: UML Diagram This UML class diagram5 represents the architecture of a back-end service for managing groomers' data in a Java application using the Spring Boot framework.

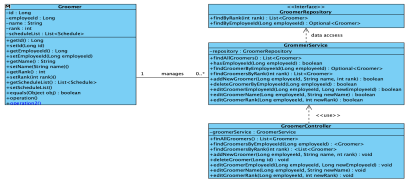

Figure 5: Groomer Management System

3.1.5 PBI # CS1105

PBI#	CS1105
Title	Displays a list of available services, including their names, types and prices
Story	As a customer, I want to be able to view a comprehensive list of available services so that I will be able to make informed decisions about the services I want to use.
Description	This PBI aims to create a feature that will allow customers to view a comprehensive list of services that are available to them. The list should include the names of the services, their types, and prices. This feature will help customers make informed decisions by providing them with a clear understanding of the services offered and their associated costs.

Design To design the PBI that displays a list of available services, including their names and types, it is needed to consider the following components and their interactions:

1. Frontend: The user interface that displays the list of available services, including names and types. The frontend will send requests to the backend to fetch this data.
2. Backend: The server-side code responsible for handling requests from the frontend, performing any necessary business logic, and interacting with the database to fetch the required data. The backend will have the following components:
    - ServiceController: A REST controller that handles HTTP requests related to services, such as fetching all available services.
    - ServiceService: A service class that contains the business logic for managing services, including fetching the list of services from the database using the ServiceRepository.
    - ServiceRepository: An interface that extends JpaRepository and provides methods for managing services in the database.
    - Service Entity: A class representing a service in the system, including fields such as id, serviceName, and serviceType.

Implementation To implement the PBI that displays a list of available services, including their names and types, it is needed to create a Service entity, a ServiceRepository, a ServiceService, a ServiceController, and update the frontend. The ServiceController handles the HTTP requests from the frontend, the ServiceService contains the business logic and interacts with the ServiceRepository to fetch the available services from the database. Here's an illustration of the required components:

1. Service entity: Create a Service class representing a service in the system. It should have fields like id, serviceName, and serviceType.
2. ServiceRepository: Create an interface that extends JpaRepository and provides methods for managing services in the database.
3. ServiceService: Create a service class that handles the business logic for managing services. This service should use the ServiceRepository to interact with the database.
4. ServiceController: Create a controller class that handles HTTP requests related to services. It should have a method for fetching all services that calls the ServiceService and returns a ResponseEntity with the list of services.
5. Frontend: Update the frontend to fetch the list of services from the ServiceController and display them, including their names and types.

Justification

1. Modularity: The system design for displaying the list of available services is broken down into distinct classes - Service, ServiceRepository, ServiceService, and ServiceController. This division fosters modularity, making the system easier to understand, maintain, and expand in the future.

2. Encapsulation: In the Service class, related data attributes such as id, serviceName, and serviceType are encapsulated, promoting data integrity and simplifying data management. This approach follows the principles of Object-Oriented Programming (OOP), promoting organized and efficient coding practices.
3. Usability: The front-end interface, which fetches and displays the list of available services, is designed to be user-friendly. The clear presentation of service names and types enhances usability, enabling customers to quickly and easily find the information they need.
4. Robustness: The implementation of validation checks at both front-end and back-end ensures the correctness of data. This robustness minimizes potential errors and contributes to the system's overall reliability.
5. Reusability: The ServiceService class is designed with CRUD operations, making it versatile and reusable. The methods in this class can be used across different functionalities within the application, promoting code reuse and reducing code duplication.
6. Scalability: The use of JpaRepository in the ServiceRepository interface allows the system to leverage the built-in database management functionalities provided by Spring Data JPA framework. This decision facilitates scalability as the system can be easily expanded to handle a larger dataset or more complex queries.
7. Maintainability: The clear segregation of responsibilities among different components (controller, service, repository) and the ability to test them independently ensures maintainability. This arrangement reduces the likelihood of issues during usage, increases system reliability, and makes the code easier to understand and modify.
8. Performance: With the ServiceController handling HTTP requests efficiently and the front-end designed for quick rendering of service data, the system is optimized for performance. This design decision results in a more responsive user interface, enhancing the overall user experience.

Visualizing System Design: UML Diagram This UML diagram helps to visualize and understand the flow of the service list retrieval process, making it easier for developers to design and implement the necessary components in the application.

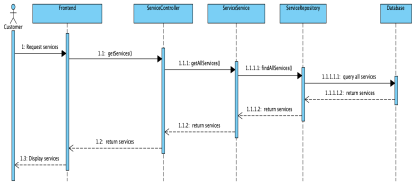

Figure 6: Service List Retrieval

3.1.6 PBI # CS1106

PBI#	CS1106
Title	Store the appointment details in the database after submission
Story	As a customer, I want to be able to submit an appointment request and have the details securely stored in the database so that I can have confidence that my appointment details are securely stored and easily managed by the manager.
Description	This PBI aims to create a feature that allows the appointment details to be stored in the database after a customer submits an appointment request. The feature will ensure that all the necessary details related to the appointment, such as the pet's information, appointment date, and service type, are recorded and stored securely in the database. The feature will enable the company to manage the appointments more efficiently and reduce the risk of any data loss or errors.

Design To design the PBI that stores appointment details in the database after submission, It is needed to create an Appointment entity, an AppointmentRepository, an AppointmentService, a UserController, and update the frontend. To realize that the UserController handles the HTTP requests from the frontend, the AppointmentService contains the business logic and interacts with the AppointmentRepository to store the appointment details in the database.

1. Appointment entity: Defining the Appointment entity with its fields.
2. AppointmentRepository: Identifying the need for an AppointmentRepository for managing appointments in the database.
3. AppointmentServicesImpl: Creating an AppointmentService for handling the business logic related to appointments.
4. UserController:Designing an UserController for handling HTTP requests.
5. Frontend: Planning the frontend updates to include a form for customers to choose a groomer and submit appointment details.

Implementation The components listed below work together to store appointment details in the database after a customer has chosen a groomer and submitted the required information. The UserController handles the HTTP requests, while the AppointmentServicesImpl interacts with the AppointmentRepository to store the appointment details in the database.

1. Appointment entity class: Create an Appointment class representing an appointment in the system. It should have fields like id, date, status, and fee.
2. AppointmentRepository interface: Create an interface that extends JpaRepository and provides methods for managing appointments in the database.
3. Update AppointmentServicesImpl: Add methods to the AppointmentServicesImpl class for creating and managing appointments.
4. Update UserController: Add a new method for creating an appointment.
5. Frontend: Update the frontend to send a request to the new UserController endpoint after a customer has chosen a groomer and entered the appointment details. This request will trigger the creation and storage of the appointment in the database.

Justification

1. Modularity: The design and implementation process divides the system into distinct components, such as Appointment entity, AppointmentRepository, AppointmentServicesImpl, and UserController. This promotes maintainability and easier understanding of the system.
2. Separation of Concerns: Each component in the system is responsible for specific tasks. For instance, the AppointmentRepository manages database interactions, AppointmentServicesImpl handles the business logic, and UserController manages HTTP requests. This principle ensures that each component is focused on its own responsibility, leading to a cleaner and more organized codebase. Encapsulation: The implementation process encapsulates the data and logic within each component, restricting direct access to their internal details. This promotes better maintainability and modularity, as changes to one component will have minimal impact on others.
3. Reusability: The design and implementation of components like AppointmentServicesImpl and AppointmentRepository follow a reusable structure, which allows for easier integration with other parts of the application and reduces code duplication.
4. Scalability: The design and implementation process follows a modular structure, enabling easier addition of new features and components in the future. This ensures that the system remains flexible and adaptable to changing requirements.
5. Maintainability: By adhering to principles like modularity, separation of concerns, encapsulation, and reusability, the design and implementation process promotes maintainability. This makes it easier for developers to understand, modify, and extend the system as requirements evolve.

Visualizing System Design: UML Diagram This UML diagram helps to visualize and understand the flow of the database storage process, making it easier for developers to design and implement the necessary components in the application.

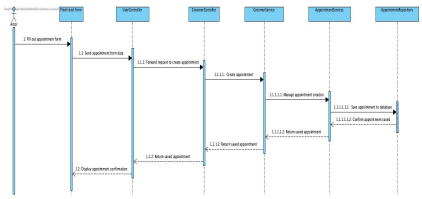

Figure 7: Database Storage Process

3.2 PBIs related to groomer's schedule

3.2.1 PBI # CS1201

Design The database schema will consist of two tables - an groomer table and a Schedule table. The groomer table will store information such as groomer ID, name, contact details, location, services provided, etc. The Schedule table will store information such as groomer ID, day, start time, end time, and availability status. The tables will have a one-to-many relationship between the groomer and Schedule tables, where an groomer can have multiple schedules, but a schedule can only belong to one groomer.

PBI #	CS1201
Title	Create a database schema to store groomer and schedule information
Story	As a customer, I want to be able to view the availability of groomers and book appointments online so that I can easily schedule appointments at a time that is convenient for me
Description	The goal of this PBI is to design and implement a database schema that can store information about groomers and their schedules. The database should be able to accommodate various types of data related to groomers such as their name, contact information, services offered, availability, and pricing. The schema should also be able to store information about the schedule of each groomer, including the time and date of appointments, the duration of each appointment, and the client name.

Implementation To implement the PBI for creating a database schema to store groomer and schedule information, we need to create two entities, Groomer and Schedule, and establish a one-to-many relationship between them.

1. Create the Groomer entity to represent a groomer within the application's domain model. This entity serves as the blueprint for storing groomer-related information, such as their employee ID, name, rank, and schedule. By defining this entity, the storage, retrieval, and manipulation of groomer data in application is facilated.
2. Create a Groomer entity to represent a groomer's schedule within the application's domain model. This entity serves as the blueprint for storing schedule-related information, such as the day of the week, start time, end time, and the groomer it is associated with. By defining this entity, the storage, retrieval, and manipulation of schedule data in application is facilated.
3. Use the Groomer and Schedule entities to display groomer and schedule information on the web page.

Justification By designing and implementing the database schema, as well as the creation of the Groomer and Schedule entities using these principles and tools, the application is more scalable, flexible, maintainable, and user-friendly, ensuring that the application can effectively manage groomer and schedule data.

1. Data Organization: By designing a database schema with separate tables for groomers and schedules, the data is organized in a way that reflects the real-world relationships between groomers and their schedules. This design allows for efficient storage and retrieval of data related to groomers and their schedules, making it easier for the application to handle complex queries and updates.

2. Scalability: Having a one-to-many relationship between groomers and schedules allows the application to accommodate multiple schedules per groomer, which is essential for managing different shifts, work hours, or service offerings. This design choice ensures that the application can scale with the growing needs of the business.

3. Flexibility: Creating separate entities for Groomer and Schedule provides a clear separation of concerns, enabling the application to manage data independently for each entity. This separation allows for easier implementation of new features, such as adding more attributes to groomers or schedules, without affecting other parts of the system.

4. Maintainability: By organizing the data in separate tables and entities, the application's code becomes more maintainable and easier to understand. Developers can work on individual entities or tables without the risk of accidentally introducing bugs or inconsistencies in other parts of the system. This modularity leads to a more robust and less error-prone application.

5. Enhanced User Experience: The implementation of the Groomer and Schedule entities allows the application to display relevant information to customers, such as available time slots for appointments and details about the groomer's qualifications. This transparency helps customers make informed decisions when booking appointments and contributes to a better overall user experience.

Visualizing System Design: UML Diagrams By using this UML class diagram8, we can effectively communicate the design and implementation of the Groomer and Schedule entities and their relationship in the application's domain model.

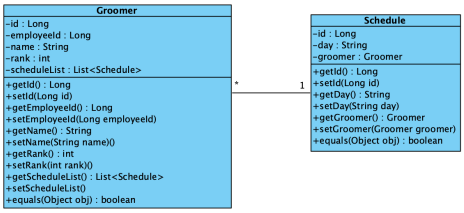

Figure 8: Groomer and Schedule Entities

3.2.2 PBI # CS1202

PBI #	CS1202
Title	Implement a back-end API to retrieve groomer and schedule data from the database.
Story	As a customer, I want to request the implementation of a back-end API that retrieves data from the esthetician and schedule database so that I can easily search and access information about the availability of different groomers and their schedules.
Description	The goal of this PBI is to create a back-end API that can retrieve data from the database created in PBI 1. The API should be designed to allow for efficient retrieval of data based on a variety of criteria, such as esthetician name, service type, or appointment date.

Design A RESTful API will be implemented, allowing the frontend to retrieve esthetician and schedule data from the database. The API will have endpoints for retrieving all estheticians, a specific esthetician by ID, an esthetician's schedules by ID, etc.

Implementation Use a framework like Spring Boot to create the API. Create appropriate controller classes and methods for handling the API requests and utilize the JpaRepository interfaces (e.g., ScheduleRepository) to perform CRUD operations on the data.

Justification By designing and implementing the API using these principles and tools, the application is more maintainable, scalable, and interoperable, ensuring a robust and efficient solution for retrieving esthetician and schedule data.

1. Standardization: REST is a widely adopted architectural style for designing networked applications, making it easy for developers to understand and work with. By adhering to the REST principles, the API becomes more consistent and easier to maintain.
2. Interoperability: A RESTful API uses standard HTTP methods and returns data in a commonly supported format (such as JSON), making it easy to integrate with various clients, including web, mobile, and desktop applications.
3. Ease of use: Spring Boot is designed to simplify the development process by providing built-in configurations, reducing boilerplate code, and offering a wide range of tools to accelerate development. Robust ecosystem: The Spring Boot ecosystem offers a rich set of libraries and integrations, which makes it easier to add new features and capabilities to the application.
4. Community support: Spring Boot has an active community, which offers extensive documentation, tutorials, and support, making it a solid choice for developing a RESTful API.

Visualizing System Design: UML Diagrams This UML class diagram9 illustrates the structure and relationships of the classes involved in implementing a back-end API for retrieving groomer and schedule data in a Java application using the Spring Boot framework.

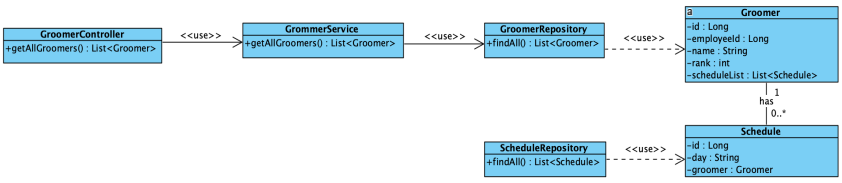

Figure 9: Groomer and Schedule Management System

3.2.3 PBI # CS1203

PBI #	CS1203
Title	Integrate the back-end API with the front-end to display groomer and schedule information.
Story	As a customer, I want to be able to view the available groomers and their schedules so that I can book an appointment for my pet.
Description	This PBI involves the integration of the back-end API with the front-end to display the groomer and schedule information. The API contains the necessary data, and the front-end needs to display it in a user-friendly way. This integration will allow customers to view the available groomers, their schedules, and make appointments accordingly.

Design

1. Front-end (Thymeleaf template): Create a Thymeleaf template that includes a table or a list to display groomers and their schedules. The template should use Thymeleaf's syntax to iterate through the list of groomers and their schedules, rendering the data on the page.
2. Controller: Implement a controller in Spring Boot application that fetches groomer and schedule data from the back-end API (using the GroomerService). This controller should handle the HTTP request for the page displaying the groomer and schedule information. It should add the fetched data to the model and return the view (Thymeleaf template) to be rendered.
3. Service: Utilize a service layer (GroomerService) that fetches groomer and schedule data from the back-end API, which will be injected into the controller. The service layer should handle the logic for communicating with the back-end API and returning the data to the controller.
4. API: Design the back-end API to provide the required data for the front-end. The API should have endpoints for fetching all groomers and their schedules. The data should be returned in a format suitable for consumption by the front-end (e.g., JSON).

Implementation

1. Create a controller method to handle the request for the page displaying groomers and their schedules.
2. Create a Thymeleaf template to display the groomers and their schedules ingormation.
3. Configure the Thymeleaf template engine in the application.properties file.
4. Ensure that Spring Boot application has the Thymeleaf dependency in the build.gradle or pom.xml file

Justification By adopting these holistic approaches below to design and implementation, which i s steeped in the principles and capabilities of Spring Boot and REST, the application's infrastructure i s robust, easily maintainable, and adept at meeting the needs of the target users.

1. Standardization and utilization: The Standardization process is rooted in the utilization of the REST architectural style, a paradigm extensively embraced for architecting networked applications. The REST principles, including the application of standard HTTP methods and returning data in widely accepted formats such as JSON, confer a level of consistency that renders the API easier to maintain. This standardization also establishes a foundation for seamless integration with diverse clients, thereby simplifying the communication between the front-end and back-end.
2. Interoperability : The API is RESTful in nature, adhering to the standard HTTP protocol, and it returns data in a universally supported format like JSON. This design choice inherently encourages interoperability, facilitating the API's seamless integration with a diverse array of clients such as web, mobile, and desktop applications. This interoperability fosters effortless data exchange and communication between different systems or platforms.
3. Ease of use and the Robust Ecosystem: The choice of Spring Boot as the development framework for the RESTful APIs is a strategic one. It streamlines the development process by o ffering built-in configurations, thus reducing boilerplate code and providing a plethora of to ols and libraries. This user-friendly nature allows developers to concentrate on business log ic rather than becoming mired in infrastructure concerns. Furthermore, the rich set of libr aries and integrations provided by the Spring Boot ecosystem makes it easier to introduce ne w features and capabilities to the application.
4. Community Support: Spring Boot boasts a large and active community of developers. This community is a wellspring of extensive documentation, tutorials, and support, consolidating Spring Boot's reputation as a reliable choice for the development of RESTful APIs. The wealth of resources and community-driven knowledge sharing ensures that developers can find solutions to challenges and stay informed of best practices, providing a reliable support network for developers utilizing the framework.

Visualizing System Design: UML Diagrams This UML class diagram10 illustrates the design and relationships between classes involved in integrating the back-end API with the front-end to display groomer and schedule information.

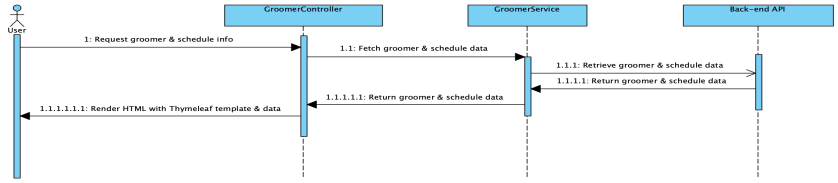

Figure 10: API Integration for Groomer & Schedule Display

4 Change Management

Requirements engineering plays a pivotal role in the field of software engineering and has a significant impact on the overall success of any software project[4]. It serves as a building block that shapes the architecture, functionality and performance of the solution. Changes are an inherent part of the software development process and occur frequently due to changing business requirements, technical issues, or user feedback[5]. These changes, if not managed effectively, can impact cost, time, and ultimately the quality of the final product. Therefore, it becomes essential to implement robust measures. These measures are designed to address changes in requirements, minimize potential dis-r uptions, and ensure that the final product meets the required quality standards, thereby increasing t he overall value of the delivered product.

4.1 Reasons and examples for requirements changing

1. New insights or ideas As the project progresses, stakeholders gained new insights or ideas that were not originally considered in the requirements gathering phase[6]. **For example**, the initial strategy was to provide users with a list of basic grooming services and some optional services, which can not show our upselling or cross-selling strategy well. However, Based on feedback and insights gained from user testing and stakeholder feedback, The final strategy involves providing u sers with a comprehensive list of grooming service packages that include basic services as well as rel ated services based on the customer's past records. This approach allows for better upselling and cro ss-selling opportunities by presenting users with tailored service packages that meet their specific nee ds and preferences.

2. User feedback Users may provide feedback during testing or after using the product in production that could require changes to the original requirements[4]. **For instance**, at the beginning of the project, only pet type and size were designed in the pet information section to meet more specific requirements. But according to user's feedback (since this is a very small-scale c lass project, the users here refer to a team of 9 developers), users have requested the ability to pr ovide additional information about their pets, such as their name and description, to ensure that th e grooming service provider is aware of any specific needs or requirements. Therefore, this feedback ha s been incorporated into the product to enhance the customer experience and meet their needs.

3. Technological problems Developing a website can be challenging and unfamiliar to the beginning developer. So, from time to time, the envisioned features cannot be implemented due to their technical limitations and the requirements have to be changed. **For example**, in this project, the appointment page has changed greatly, the intial page was made by using HTML, CSS, JavaScript, and jQuery. However, the final page just use HTML, CSS. Because, due to the t chenical limitation, it is hard for me to fetch the data in JavaScript. Additionally, the deadline was a pproaching, so the requirements have to be changed. But, all the function implemented successfully fi nally.

4.2 Handling requirement changes in a Scrum project

In a Scrum project, requirement changes are inevitable as the project progresses. Agile methodologies like Scrum emphasize flexibility and adaptability, allowing teams to embrace changes and deliver a high-quality product that meets users' needs. Here are some ways requirement changes can be handled in a Scrum project:

1. Embrace change: Scrum teams should acknowledge that requirements can change over time and be open to making adjustments as needed. Agile processes encourage continuous improvement and adaptation, which helps ensure the final product meets user expectations.

2. Prioritize changes: When new requirements arise, the Product Owner should assess their importance and prioritize them in the Product Backlog. This will help the team focus on high-priority tasks and ensure that the most critical changes are addressed first.

3. Break changes into smaller tasks: In order to manage changes effectively, break t hem down into smaller, manageable tasks or user stories[7]. This will make it easier for the team to es timate the effort required and plan for the implementation of these changes in upcoming sprints.

4. Collaborate with stakeholders: Engage stakeholders in discussions about the changes and their impact on the project[7]. This collaboration will help to create a shared understanding of the new requirements, identify potential risks, and determine the best course of action for implementing the changes.

5. Reevaluate estimates and timelines: Requirement changes may affect the project's scope and timelines. The team should reevaluate their estimates and adjust their plans accordingly to ensure that they can deliver a high-quality product within the given time frame.

6. Communicate changes: Keep the entire Scrum team informed about requirement changes, their impact on the project, and any adjustments to the plan. This transparency will help the team to understand the reasons behind the changes and adapt their work accordingly.

7. Review and learn: Scrum teams should regularly review the impact of requirement changes on the project and learn from their experiences. During Sprint Retrospectives, the team can discuss any challenges they faced in managing changes and identify ways to improve their processes in future sprints.

5 Legal, social, ethical, and professional issues

In the development of an appointment page for a pet grooming service, several legal, social, ethical, and professional issues have been taken into account. This discussion will outline these considerations and propose possible solutions for each.

Data Privacy and Protection Considering the appointment page collects personal information such as pet names, types, sizes, descriptions, and the groomer's schedule, it is essential to ensure compliance with data protection regulations like GDPR or CCPA. The platform must provide clear and transparent privacy policies, obtain user consent for data collection, and ensure data is securely stored and managed.
Possible Solution: Implement a secure and easy-to-use authentication and authorization system to ensure that only authorized users can access the schedule information. The system will have a user authentication mechanism requiring users to log in to access schedule information and an authorization mechanism that restricts access to certain pages or functions based on user roles.

Accessibility and Inclusivity To ensure the appointment page is accessible to a diverse range of users, including those with disabilities, it must follow web accessibility guidelines, such as the Web Content Accessibility Guidelines (WCAG).
Possible Solution: Design the appointment page with an inclusive user interface, considering color contrasts, font sizes, and keyboard navigation. Provide alternatives for visual content, such as alt-text, and test the page with accessibility tools to ensure compliance with accessibility standards.

Fair and Ethical Treatment of Customers As the appointment page is responsible for scheduling grooming services, it must ensure that customers are treated fairly and without discrimination. The scheduling algorithm should be transparent and unbiased.
Possible Solution: Ensure that the scheduling algorithm is transparent and does not prioritize specific users based on factors unrelated to appointment availability, such as pet size or service type.R egularly audit and review the algorithm for any potential biases or unfair treatment.

Transparency and Clarity in Service Offerings It is crucial to provide accurate and c lear information about the different service types (basic service and service package) available to c ustomers, including pricing and what each package entails.
Possible Solution: Clearly list the services included in each package, along with their respective costs. Use simple language and visual aids to help users understand the offerings better.

By addressing these legal, social, ethical, and professional issues, the appointment page for the pet grooming service will provide a user-friendly, accessible, and secure platform for scheduling grooming appointments while ensuring fair treatment and professional interactions between customers and groomers.

6 Conclusion

In conclusion, the development of the service reservation system has provided valuable insights and lessons for designing and implementing a software project. By utilizing the Model-View-Controller (MVC) architecture, Repository pattern, Service Layer, Configuration Files, Static Resources, and Templates, the system has been designed for efficiency, maintainability, and scalability. For future projects, it is essential to continuously review and refine the architectural design, ensuring optimal performance. Moreover, maintaining up-to-date and comprehensive documentation is crucial for facilitating future modifications and enhancing team collaboration. Emphasizing reusability and modularity in the development process will contribute to a more structured and professional software product.

References

[1] P. L. Ayunda and E. K. Budiardjo, "Evaluation of scrum practice maturity in software development of mobile communication application," in *2020 3rd International Conference on Computer and Informatics Engineering (IC2IE)*, pp. 317–322, 2020.

[2] V. T. Faniran, A. Badru, and N. Ajayi, "Adopting scrum as an agile approach in distributed software development: A review of literature," in *2017 1st International Conference on Next Generation Computing Applications (NextComp)*, pp. 36–40, 2017.

[3] G. Rodríguez, I. Gasparini, A. Kemczinski, and A. Veloso de Matos, "Students' perception of scrum in a course project," *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, vol. 16, no. 4, pp. 329–336, 2021.

[4] A. A. Alsanad, A. Chikh, and A. Mirza, "Multilevel ontology framework for improving requirements change management in global software development," *IEEE Access*, vol. 7, pp. 71804–71812, 2019.

[5] S. Jayatilleke and R. Lai, "A systematic review of requirements change management," *Inf. Softw. Technol.*, vol. 93, p. 163–185, jan 2018.

[6] N. Saher, F. Baharom, and O. Ghazali, "Requirement change taxonomy and categorization in agile software development," in *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)*, pp. 1–6, 2017.

[7] N. Baruah, "Requirement management in agile software environment.," *Procedia Computer Science*, vol. 62, pp. 81 – 83, 2015.