CPT202 Assignment 1
Group Report for Software Engineering Group Project

2022/2023 Semester 2
Project B

May 5, 2023

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`

Group number: 9
Submitted by Guanyuming He, 2035573
Group Members:

1. Kairui Bai, 2039195
2. Yuchen Feng, 2033486
3. Guanyuming He, 2035573
4. Kaijie Lai, 2034675
5. Zhen Ma, 2034590
6. Jing Yin, 2034355
7. Songyi You, 1930422
8. Hanyu Zhang, 2034628
9. Yumeng Zhang, 2035352

[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

1.1 Problem Statement

In today's digital world, most customers want to be able to schedule their pet grooming appointments online without speaking to a live person for convenience. As a manager of a pet grooming business, if you do not offer online appointment scheduling for your customers, it is highly possible that you will lose a huge potential source of income.

1.2 Aims of The Project

This project aims to develop a pet grooming appointment website for a pet shop, not only enabling customers to save time by sending their schedule to the shop automatically but also allowing the manager to keep track of every client's information and preferences, then deliver the best possible service.

1.3 Project Scope

This website is designed for the manager of the pet shop and all the people who want their pets groomed in this shop. It can be accessed 24 hours a day. Functionalities for customers:

1. Register an account, log in
2. View pictures of the shop's appearance.
3. View all the groomers, services, and discounts with detailed descriptions.
4. Make and cancel appointments, view appointment history with status provided.
5. View and make comments on each service.
6. View FAQs and help documents to find answers to questions related to grooming and appointment.

Functionalities for the manager:

1. Log in to the system.
2. Maintain system master file, selling strategies, and pictures of shop appearance.
3. View received appointment orders.
4. View business statistical reports.

Functionalities NOT included in this system:

1. Pay for the appointed services or ask for a refund.
2. Apply for a job in the shop.
3. Send messages between the manager and the customer.

1.4 User Characteristics

The primary users of the system are pet owners who own dogs or cats and require grooming services for their pets and appreciate the ease and efficiency of the online appointment booking experience. The system is intended to cater to a broad demographic of pet owners, regardless of age, gender, or location, who may have busy schedules, prefer to avoid the hassle of waiting in line, or want to avoid the frustration of traveling to the salon and discovering that no appointments are available. In addition, the platform is designed to enable managers to efficiently operate and manage the system.
The website features a clean and intuitive interface that enables all users, regardless of their technical expertise, to easily book appointments. The system's user-friendly design ensures that even non-professionals can easily use the platform.

1.5 Assumptions and Dependencies

1.5.1 Assumptions

The following assumptions have been made regarding the pet grooming service provider and their use of the web-app pet grooming system:

1. The service provider has the necessary hardware and software requirements to run the system effectively.

2. The service provider is willing to allocate the necessary resources to integrate the system into their existing workflow.

1.5.2 Dependencies

To ensure the reliability and stability of the appointment system, the following dependencies must be considered:

1. The running status of the server: It is assumed that the server is functioning properly and can support the system.

2. Network connectivity between the server and clients: It is assumed that there is a stable and reliable network connection between the server and the clients using the system.

3. The stability and maintenance of the system's hardware and software updates: It is assumed that the system's hardware and software are well-maintained and updated regularly.

4. A stable internet connection from the user's side: It is assumed that users have a stable internet connection that can support the booking process.

1.6 Project Risks

During the process of development, the Ali Cloud server was used to warn that some of the servers were under attack or being used to conduct mining and these servers had been shut down to ensure security. As a result, a strong and complicated password had been utilized to prevent attacks.

2 Architectural Design

This system is a web-based system handling pet-grooming appointments. In this section, functional and non-functional requirements will be examined and their impact on the architecture will be discussed. Then, the architectural design decisions made according to them will be explained.

Functional requirements related Overall, the functional requirements of the system exhibit the characteristics of those of a *transaction processing system*.

It is built around pet grooming *appointments*. Users *request* information about the appointments they make and information regarding how they make appointments, like service type and groomer data, from a database. Concurrently, they request to *update* the database when they make or modify their appointments.

Any side requirements introduced generally also agree with these properties. For example, a user's giving reviews or comments on one of his appointments is essentially updating the database as well.

Non-functional requirements related The system architectural design depends closely on the following non-functional system requirements.

- *Performance* The system is not performance-critical. As long as a user does not feel obvious lag during his actions then the system can be accepted.
- *Security* Security is an important requirement, because a security breach could result in serious loss or damage to a user's property, like their pets or money. The system must provide protection systems protecting the users' data from attacks and leakage.
- *Safety* Safety measures are crucial. On software failure, the system must provide additional support for a safe shutdown so that all the data in the database remains unchanged.
- *Availability* Availability is not an important requirement for this system. The system does not need superfluous components for the update or replacement of others.
- *Maintainability* This can be important but not necessary at this stage. The system can be designed with change-ready components wherever appropriate.

2.1 System Architecture

2.1.1 Architectural views

Here which perspectives or views that are used for the architectural design are discussed.

The system is designed through these fundamental views from Krutchen's 4+1 model [1].

- *A logical view* In this view many system requirements are abstracted into object classes, which also can act as entities of the database.
- *A process view* How processes inside the system interact with the system and the environment are investigated in this view, which helps the assessment of the non-functional requirements.
- *A development view* How the system development is divided and how the tasks are allocated to individual developers are discussed. According to the project specification, the allocation of tasks ensures that everyone has a touch on every aspect, while they may have some main focus.

2.1.2 Architectural pattern

The system uses the MVC architectural pattern. It is one of the most prominent patterns for a web-based system like this system, as [2] points out. In addition, it is good for the system for the reasons below.

MVC divides the system into three components, so the interaction and presentation are isolated from the data. The data is the most stable part of this system while the presentation can change drastically during the development. Therefore, this separation done by MVC enables developers to implement the other components while leaving the data unchanged.

2.1.3 Architecture details

In this part, how the MVC pattern is adjusted to fit the requirements of the system is explained, with the results shown in the text and in Figure 1. There, in addition to a basic MVC structure, more facilities are added.

Security The security system for data access management can be put inside the componentss. However, this jeopardizes the independence of the components since a change in security may involve changes in more than one component.

Therefore, the decision is to use a standalone security layer provided by Spring Boot outside of the components, as shown in the figure. This way the independence of components are preserved, the security may be better than creating security methods from the ground because Spring Boot is a powerful and stable framework, and efforts are reduced because a component is reused.

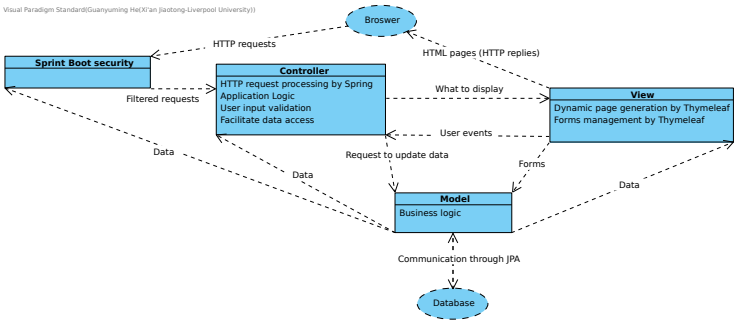


Figure 1: MVC architecture diagram

Caching or queuing server Caching or queuing servers are not used in the system.

This decision is made considering the nature of the functional requirements. The database should not be updated until a appointment making process is completed and should be updated as soon as possible after the completion so that the manager can be notified. The delay introduced by caching or queuing does not satisfy the purpose. In addition, the size of data communicated is not large for appointment making and checking – the core requirements, since an appointment only includes several short strings and integers in the database.

As for non-crucial data like images in user reviews, cookies can be used for caching without the need for a server.

2.2 Software Modules

The system is modularized in two levels. At the first level, methods and variables are grouped into Java classes. At the second level, classes are grouped into bigger modules by various factors. The class diagram and the module diagram are shown in Figure 2. The note in each diagram explains the things that are not obvious in the graph. The relationships between classes are clearly and comprehensively demonstrated.

Services of entities are methods of the entity classes. Non-entity classes that are solely for the Spring Boot framework like controllers do not appear in the class diagram.

2.2.1 Justification

Class grouping First, according to the fundamental requirements, core functions and data are grouped into these important classes:

`Account`, `Appointment`, `ShopManager`, `SellingStrategy`, `MasterFile`, `StatisticalReport`.

Then, auxiliary functions and data for them are grouped into some other classes. In addition, functions and data for other requirements are grouped into the rest

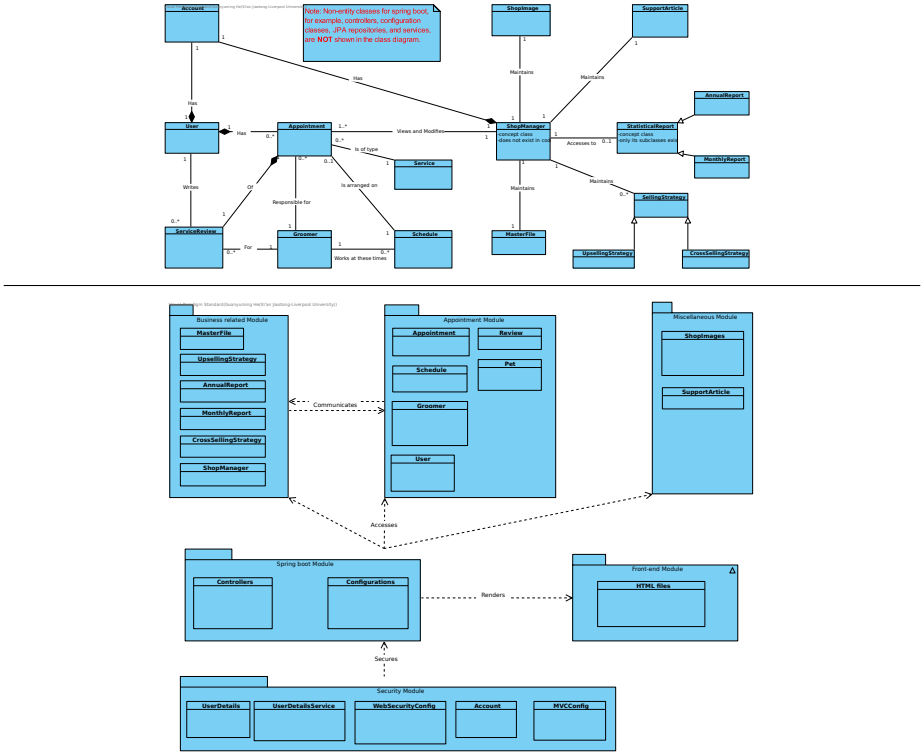


Figure 2: Class and module diagrams

classes. About non-entity web classes, their methods correspond to specific HTTP requests or Spring framework methods. Their fields are created for the methods. There are some big changes during the implementation, however, that, first, all functions in `ShopManager` are moved to a dedicated controller class and `ShopManager` is removed; second, due to the specialty of different statistical report types, the super class `StatisticalReport` is removed its subclasses stand alone themselves.

Not only requirements are considered for grouping classes. Cohesion and coupling are also estimated so that each class represents a quite standalone group of functions. Furthermore, factors like how likely a function is to change are also taken into account to decrease development costs.

Module grouping Most of the factors that affect how modules are formed are the same as those for forming classes. However, the weights of the factors are different. Modular principles like low coupling and high cohesion are more important in forming modules than in forming classes. Apart from that, classes are carefully put into modules in a way that promotes the team collaboration. Functions that require roughly the same skill set are tried to be put in the same

module. Note that everyone still has to touch every aspect of the project through inter-module communications and modules only cover the back-end workings so everyone has to touch a bit front-end matters.

The *Appointment module* deals with appointment-related works, including its auxiliary classes like `Review` and `Pet`. The *Business-related module* is a collection of the matters the shop manager cares about. The *Security* module holds all classes that take effect in securing the product. The *web classes* module has all the framework related classes. The *miscellaneous module* holds all other standalone auxiliary or utility classes.

2.3 High-level Database Design

Because the database is generated from Java classes by JPA, the ER diagram in Figure 3 resembles the class diagram in 2. However, not every Java class has a corresponding table (entity) in the database. Moreover, the relationships are adjusted based on the associations in a way that problems like chasm traps and fan traps are avoided in the diagram.

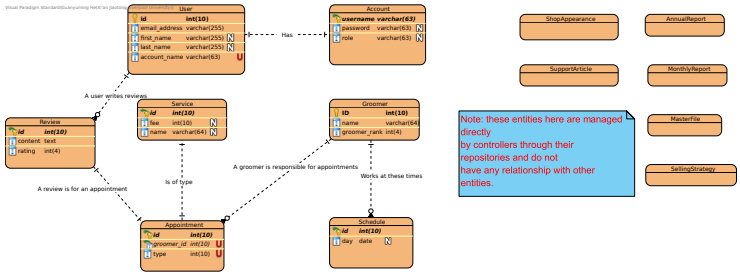


Figure 3: Entity-Relationship diagram

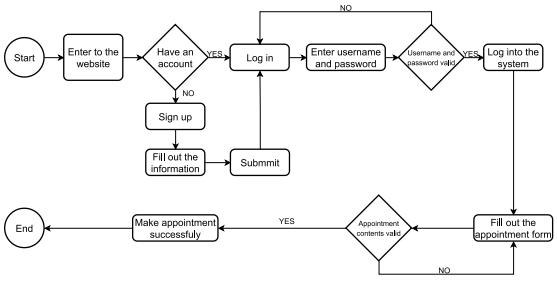
3 Software Design

There are requirements that has impacts on the above architectural design:

Security: If the system handles sensitive data or processes transactions, it may require a design that includes robust security measures such as encryption, access controls, and secure communication protocols.

Maintainability: If the system needs to be easily maintained and updated, it may require a modular design that separates different components and makes it easy to swap out individual modules as needed.

3.1 High Level Design



3.2 Software Support Services

3.2.1 Database-related services

This involves managing the data storage and retrieval for the system. Common services include database design, creation, maintenance, and backup. Popular database management systems include MySQL.

3.2.2 Security-related services

The application is secured by Spring Boot security, which enables and manages the authorization and the authentication of our product. Because the system will be utilized by both the customers (the users) and the shop manager, it is essential that support for access control is provided. Spring Boot security is great for this purpose. HTTP requests are secured and an authenticated user cannot access most webpages. In addition, a user of some privileges cannot access pages that require other privileges.

3.2.3 Webpage navigation related services

This service helps users navigate through the system's web pages. It includes website design, layout, and user interface (UI) design.

3.2.4 Hosting-related services

This service provides the infrastructure for the system to run on the web. Hosting-related services include cloud hosting providers like, as well as managed hosting providers.

3.3 Coding Structure and Convention

The Coding Structure and Convention section of this report details the standard structures and coding conventions used in the development of the service reservation system. This section explains the importance of documenting coding guidelines and conventions and provides recommendations for future development teams.

3.3.1 Standard Structure

The service reservation system was developed using the Model-View-Controller (MVC) architecture, Repository pattern, Service Layer, Configuration Files, Static Resources, and Templates.

- MVC Architecture: The MVC architecture separates the application into three interconnected parts - Model (data), View (UI), and Controller (logic). The Model represents the data access layer, View represents the presentation layer, and Controller represents the business logic layer.
- Repository Pattern: The Repository pattern provides a separation between the data access layer and the business logic layer. The Repository pattern abstracts the data access code and provides a simplified interface to the business logic layer. In the service reservation system, the Repository pattern was used to interact with the database through JPA.
- Service Layer: The Service Layer provides a layer of abstraction between the Controller and the Repository. It contains business logic and acts as an intermediary between the Controller and the Repository. In the service reservation system, the Service Layer was responsible for processing data, applying business rules, and calling the appropriate methods in the Repository.
- Configuration Files: Configuration files contain various configurations for the project, such as database configuration, application properties, and security settings. In the service reservation system, the configuration files were stored in the resources folder.
- Static Resources: Static resources are files such as CSS, JavaScript, and images that are used to enhance the user interface of the application. In the service reservation system, these files were stored in the resources/static folder.
- Templates: Templates are HTML files that contain the markup for the UI. In the service reservation system, the Thymeleaf template engine was used to render HTML templates. These files were stored in the resources/templates folder.

3.3.2 Coding Convention

The Google code style was used in the development of the appointment system. The Google code style is a set of guidelines for writing code in the Java programming language. It is designed to improve the readability and maintainability of code and to make it easier for developers to collaborate on large codebases.
The following guidelines were followed in the appointment system:

- Naming Conventions: Variable and method names followed camel case naming convention. Class names followed Pascal case naming convention.

- Code Formatting Guidelines: The code was formatted using the Google Java Style Guide. This includes two space indents and no tabs, with a line length of 100 characters. The style guide also specifies the use of whitespace and the placement of braces.
- Comments: JavaDoc comments were used to document methods and classes. Inline comments were used sparingly, and only when needed to clarify the code.

The benefits of using a consistent coding convention include improved code readability, easier code maintenance, and better collaboration among team members. By following the Google code style, the codebase is more consistent and easier to navigate, making it easier for team members to read and understand each other's code.

3.4 Software Configuration and Production Environment

The section details the configuration of the software used in the appointment system and the production environment in which it is deployed. And the part provides an overview of the software stack, deployment architecture, and infrastructure used in the system.

OS/Software	**Version**
Linux	Ubuntu 20.04
Java	17.0.6
MySQL	8.0.32
tmux	3.0

The production environment is hosted on an Aliyun server running Ubuntu 20.04 LTS. MySQL 8.0.32 is used for the database. The server is configured for automatic updates to the operating system and software packages to ensure that the latest security patches are applied.

The system is deployed using a Docker container, providing easy portability and scalability. GitHub Actions are used for continuous deployment, automatically building and deploying code changes from the GitHub repository to the production environment.

The database is hosted on a separate instance and regular backups are performed to ensure data integrity.

Overall, the software configuration and production environment are designed for reliability, scalability, and security. The use of the latest software versions and automation tools ensures that the system is up-to-date and that changes can be quickly and safely deployed to the production environment.

4 Software Testing

The product is tested against unit testing, integration testing, and acceptance testing. In this section, how they are conducted will be discussed with some examples shown.

4.1 Testing procedure

After the completion of any PBI, it is required that its author has his unit tests executed. When any branch is merged into the master branch or whenever the master branch receives a new commit, the entire unit test suite as well as the entire integration testing suite will be executed to make sure everything still works.

After each Sprint, acceptance testing will be conducted where the programmers gather to run and inspect the whole product, checking if it meets the acceptance criteria of all the PBIs implemented in that Sprint.

Unit and integration tests are written as JUnit and Spring Boot tests, so that they can be executed automatically. Each module has a folder inside `src/test`, where there is a `unittest` folder if the modules needs unit testing, and a `integrationtest` folder if it needs integration testing.

4.2 Unit testing

Important business logic of the product is tested by unit testing, where each method is tested several times, each time for a different scenario. For example, to test if class `Account` can check the rules of username, password, and role in its constructor, these test methods are written to test the constructor:

- `testConstructorAllCorrect()`
- `testConstructorInvalidUsername()`
- `testConstructorInvalidPassword()`
- `testConstructorInvalidRole()`

A more detailed example of unit testing is shown in Table 1.

Test name	`testConstructorInvalidPassword()`		
Precondition	None		
Inputs	(param password) Too long	too short	contains illegal characters
Outputs	None		
Expected	an exception	an exception	an exception
Actual	an exception	an exception	an exception
Test process	with JUnit 5		

Table 1: An example of unit testing

4.3 Integration testing

The two major parts that are tested by integration testing are, communication with the database, and the HTTP request handling.

For communication with the database, because in the product, Spring services that involve some logic are built upon JPA repositories, it involve not only simple interaction with the database, but also some business logic that are not and cannot be tested by unit testing.

For HTTP request handling, both security configurations like if some URLs can be accessed by a user with specific role and some controller functions are testes.

Table 2 shows two examples of the integration testing, one for database and another for HTTP requests.

Test name	`testFindAllGroomersTwoGroomer()`
Precondition	When the database already stores the information of more than groomers.
Inputs	`groomerService.findAllGroomers()`
Outputs	A list of groomers found
Expected	A list of all groomers that are stored in the database.
Actual	A list of all groomers that are stored in the database.
Test process	with JUnit 5

Test name	`testRestrictedPagesWhenNotAuthenticated()`
Precondition	When the URLs used in the integration testing are configured to require authentication. That is, not everyone can access them.
Inputs	`HTTP GET requests for the URLs`
Outputs	HTTP responses.
Expected	HTTP 302 for all URLs accessed (which indicate that the user will be redirected to the somewhere else, namely the login page).
Actual	HTTP 302 for all URLs accessed (which indicate that the user will be redirected to the somewhere else, namely the login page).
Test process	with JUnit 5

Table 2: Two examples of integration testing

4.4 Acceptance testing

After each Sprint, the whole product from the master branch will be run, and the all web pages involved in the PBIs completed in that Sprint will be accessed in a browser, as if a real user is doing it. The acceptance criteria will be checked one by one for each PBI.

Figure 4 is a photo of our conducting the acceptance testing.

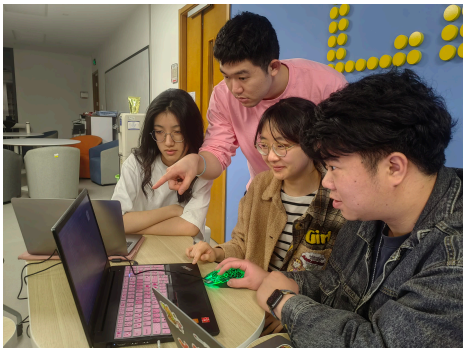


Figure 4: Example of acceptance testing

Appendices

Appendix A Member contribution

All members agree that each one contributed equally to the project. There is no need for a table calculating the contributions.

References

[1] P. B. Kruchten, "The 4+1 view model of architecture," *IEEE software*, vol. 12, no. 6, pp. 42–50, 1995.

[2] I. Sommerville, *Software Engineering, 9/E*. Pearson Education India, 2011.