Pet Diaries

Final project Report

Project Group 51
Ankita Manjrekar
Anne Maria Vasu
Gayatri Mestry

I. Implemented Features

Feature ID	Title	Requirement ID reference
001	Create a website for Pet Diaries	Business Requirement
002	System will have three categories of actors - Admin, owner, caretaker	Business Requirement
003	Create a chat facility for owners and caretakers	Business Requirement
004	Creating 'virtual money" to compensate for caretaking hours	Business Requirement
	Managing database that stores user information and processes	Business Requirement
005	Validate credentials and allow login if passes. If validation fails display corresponding error message to user	Use Case - 01
006	Validate user inputs in all login and registration forms	Use Case – 01 Use Case - 02
007	Implement account lockout on 3 invalid attempts to login	Use Case - 01
008	Lock user account on three invalid login attempts with wrong password	Use Case – 01

009	Send recovery email with unique token to authenticate and enable valid password recovery	Use Case - 01
010	Can create a profile with username and password	Use Case - 02
011	Choose a role as owner or caretaker during profile creation	Use Case – 02
012	Owner dashboard, caretaker dashboards and admin dashboards are displayed based on decisions taken by code at runtime.	Use Case-01
013	Give preference of pets	Use Case – 02
014	Choose number of pets and type of pets after initial login	Use Case – 02
015	Add phone number, address and email address after initial login	Use Case – 02
016	Block request raising and accept request till pet information/pet preferences and contact information have been set by user using user dashboard	Use Case - 02
017	User profile are populated as per information set in user registration	Use Case - 01
018	Checks the profile activity in database and marks a profile for deletion	Use Case – 07
020	Admin Sends an email to the owner/caretaker to activate their profile and marks the expiry of it up to 2 days from the email sent	Use Case - 07
021	Database is updated for with latest login timestamp if the user logs in.	Use Case 01
022	Admin updates the database to remove users from expiry list if they have logged in after expiry mail was sent, but before profile expired.	Use Case - 07

023	Admin deletes the user profile from database if users have not logged in for 2 days after expiry mail was sent out.	Use Case - 07
024	Requests will turn up in caretaker dashboards based on the pet preferences they had previously set while creating the profile.	Use Case - 07
025	Owner can generate new requests	Use Case -04
026	Owners can view caretaker profiles	New use case introduced to implement design logic
027	Owners will get a list of Suggested Caretakers based on history of Caretakers who have taken care of Owner's pets.	New use case introduced to implement design logic
028	Owners can edit previously raised requests.	Use Case – 04
029	Caretakers respond to requests that turn up in their inbox based on preferences and location set while setting up profile	Use Case - 04
030	Admin can authenticate new user requests	Use Case - 05
031	Give feedback to Caretaker	Use Case -06
032	Admin can selective delete requests raised by Owners	Use Case-04
033	Admin can view activities between all users by monitoring requests coming up in the system	Use Case-04
034	Admin notifies owner that their request has been accepted	Use Case -05
035	Admin can selectively deactivate User accounts	Use Case-05

036	Caretakers can accept requests raised by owners	Use Case - 03
037	Owners can select caretakers after viewing their profiles	Use Case-03
038	Generate temporary chat box	Use Case-03
039	Open chat box	Use Case-03
040	Send message in chat box	Use Case-03
041	Accept agreement	Use Case-03

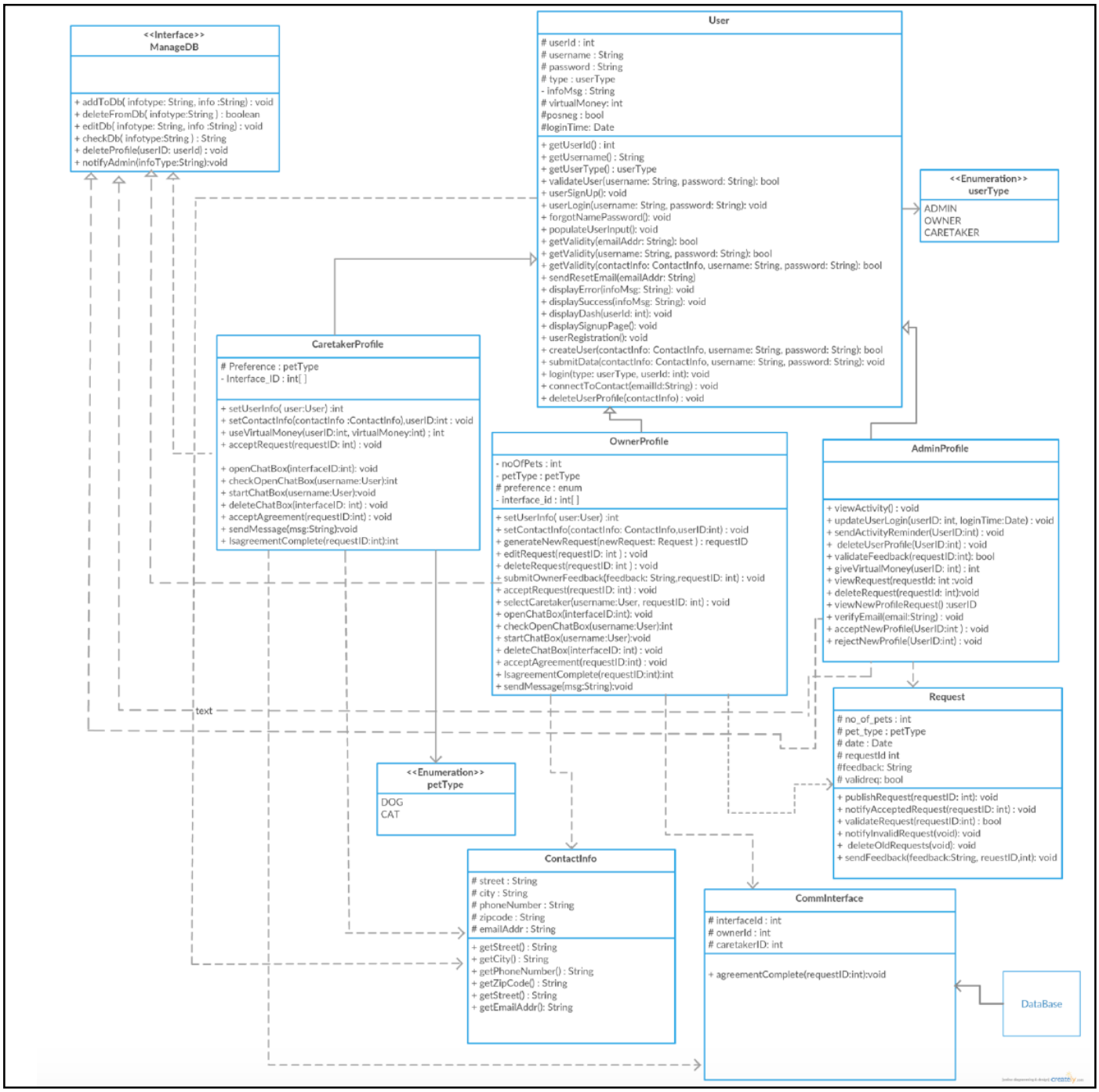
II. Features that were not implemented

001	Delete previous chat box	UR-023	This was not implemented since application's performance was not deterred by increasing chat windows.
002	Block Users based on negative feedback	UR-027	Users can be selectively deleted by admin user. Also, real-life systems don't necessarily block user based on feedback. Furthermore, the user could always create an account with new account details. Hence, this case was dropped.
003	Owner sets pet preferences	UR-024	Owner fills up Pet info on login, whereas Caretakers fill up Pet preferences to indicate Pet they wish to take care of. It was decided that the current system would clearly segregate between Owners and Caretaker, and thus this use case was

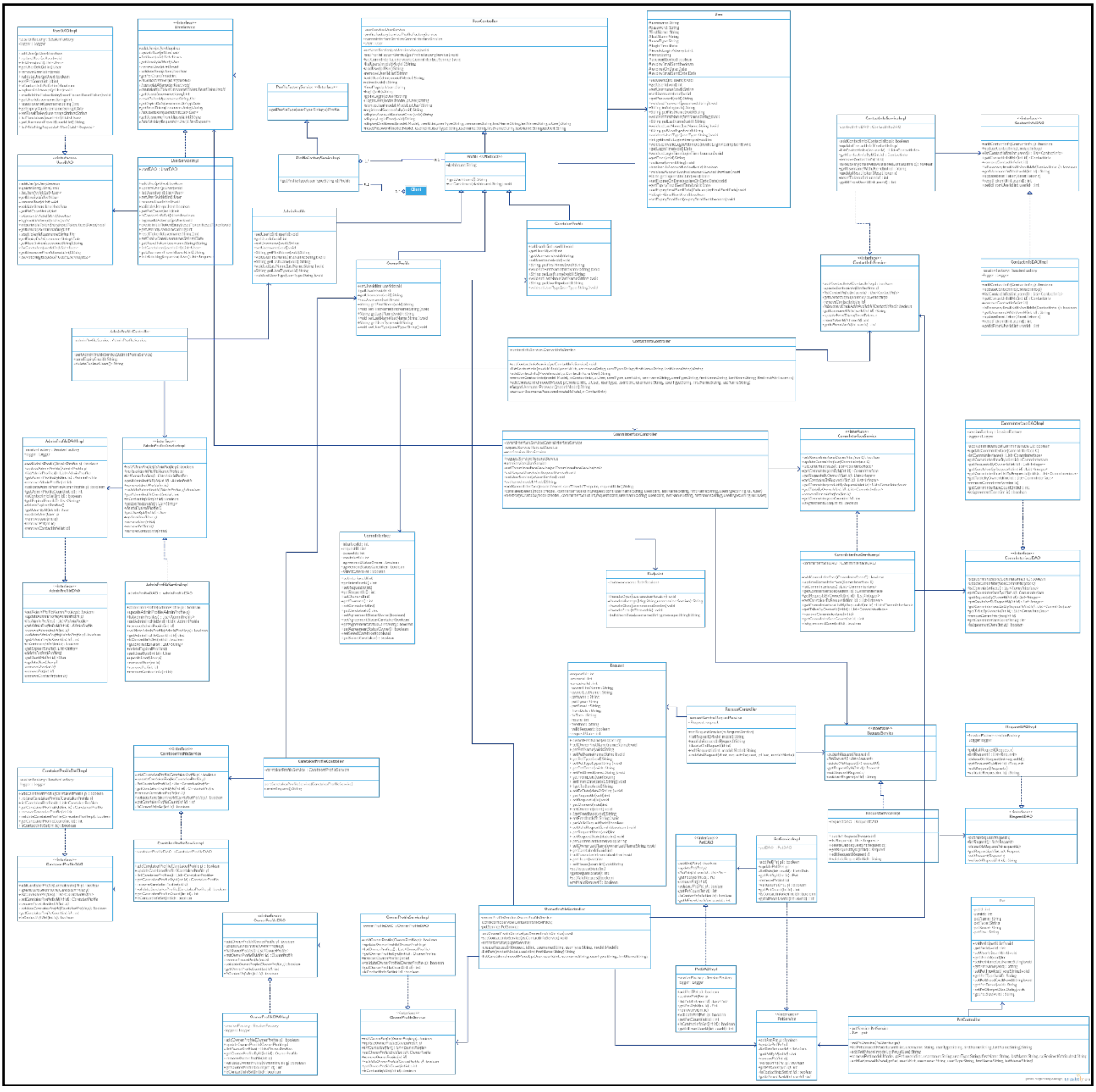
			dropped.

III. Class Diagram Description

1) Part 2 Class Diagram



2) Final Class Diagram



(Note: A PDF version of the above class diagram has been provided – Part6_ClassDiagram.pdf

New Updated Changes :

→ What changed from Part 2's design pattern and why?

- While the underlying idea and features of the project stayed the same, quite a few changes were introduced in implementation.
- The reasons for these changes are mentioned below in increasing order of importance:

1. SPRING and HIBERNATE

- The initial design idea was done before completing an initial round of experimentation with Spring and Hibernate.
- However, implementation of Spring and Hibernate completely changed the project team's basic implementation idea.
- Together, these two utilities brought in a huge amount of power via the use of beans, Controllers, Services and DAOs.
- Spring achieved loose coupling through dependency injection and interface based programming.
- Spring also provided new ways to expose services of objects such Communication Interface, Request and Admin to presentation layer components which is one of the main reasons for the change in the class diagram

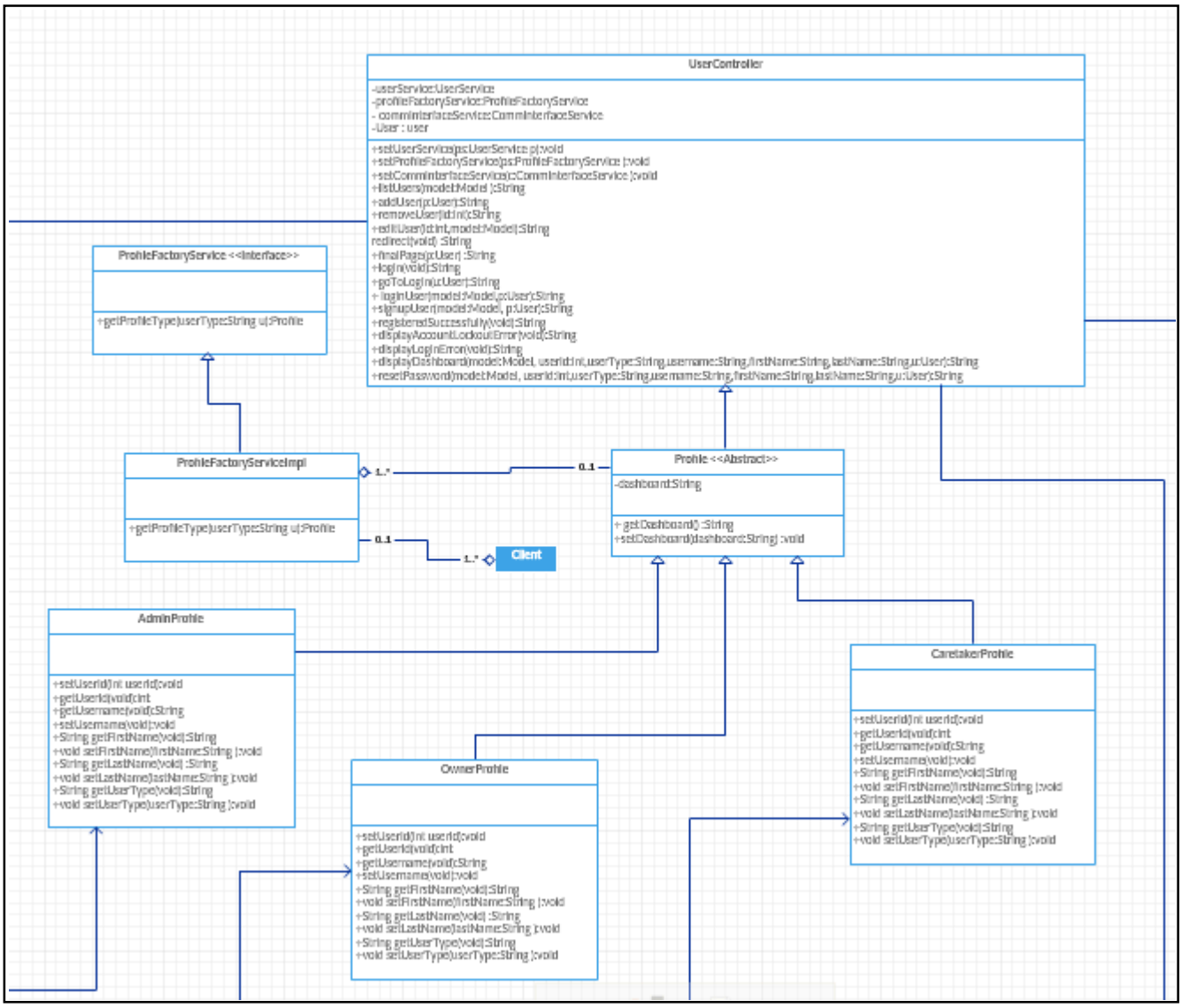
2. Changes in a design decisions made during Part 2

- While the design decisions made in Part 2 were extremely helpful in kick starting the project, coding had not yet commenced.
- As coding gathered pace, some changes in the initial design came into picture.
- For example, Pet class was added and enumerations were removed.
- Furthermore, design patterns were considered and added, and refactoring was done as the development progressed. Thus, the class diagram changed from Part 2.
- As can be seen in Part 2's class diagram, implementation of the database is not clearly shown.
- As Hibernate saved a lot of time that would have otherwise been spent in creating redundant SQL queries.

IV. Design Patterns implemented

The Factory Design pattern was implemented for the login use case.

- The main idea behind this was that there are currently three types of users in the System – Admin, Owners and Caretakers.
- However, more users such as Moderators, Advertisers etc. can be added to the System.
- Therefore, it did not make sense to change the User Controller each time a new User is added.
- Instead, a Profile Factory method would provide the User Controller with the object of Admin, Owner, Caretaker classes, and the User controller would facilitate navigation to the required dashboard using this object rather than having a sequence of if-else conditionals to determine which dashboard to navigate to.
- The implementation of Factory Design Pattern is shown below.



Design Pattern(s) that could have been implemented

Proxy: Proxy pattern was initially chosen for accessing the database. However, Hibernate proved to be extremely efficient for DB queries, and thus the Proxy design pattern was not implemented. Since Pet Diaries deals with a lot of DB queries, Proxy would still appear as a good design pattern for this implementation.

Fly Weight: Pet Diaries has not displayed any images for now. However, Fly Weight can be used as part of future to improve the UI and enhance user experience.

V. New Lessons Learnt

- To be able to successfully create and design system, it is critical to clearly understand requirement of project and role of each actor involved in project.
- With requirements in mind it is easier to develop understanding of dependencies between various functionalities of project.

- The concept of design patterns helped in collaborating with and optimizing the way the classes interacted with each other.
- The use of Spring and Hibernate helped in getting an insight into the internal workings of how the two frameworks interacted to gel the model view controller with the Database framework.