EE2010 Software Engineering
Report PDF
Aymen Mtibaa

Software Requirements Specification

Introduction

This document presents the documentation of software requirements specification(**SRS**)for a program that will be used to *adopt pets*. This project is a product of Group 1 of EE2010 for the 2024/25 academic year. The document has a list of functional and non-functional requirements of the program and also a USE-CASE diagram. The structure of this document is based on the IEEE Recommended Practice for Software Requirements Specification [1].

1.1 Terminology

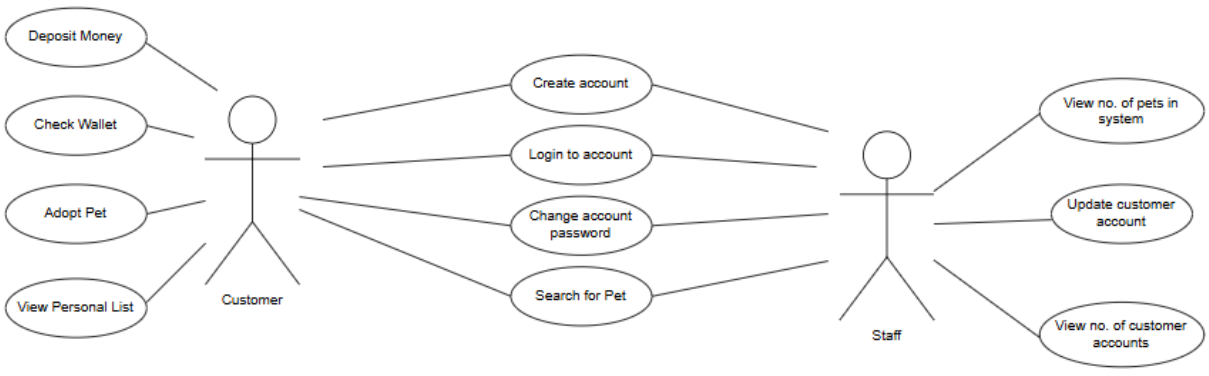
Pet	A domestic animal kept for companionship or leisure that people can adopt on our program.
System	The network that is responsible for storing all data in the program.
Customer	A person who adopts pets from our program.
Staff	A person who is responsible for updating the pet database on our program.
Money	Medium exchanged for pets. It is inputted from a user's card onto the program and into their wallet.
Account	An arrangement for using the program service by either a Customer or Staff member. It contains details of the person involved and allows access to specific aspects of the

	program.
Type	The kind of animal a pet is.
Breed	A stock of animals within a species having a distinctive appearance and typically having been developed by deliberate selection.
Name	A word or set of words by which a Pet, Staff or Customer is known, addressed, or referred to.
Age	The length of time a Pet has existed.
Gender	The male or female sex of a Pet.
Wallet	Location where Money is stored on the program.

Overall Description

The system provides adoption services to customers who are looking for pets. It has a database of numerous domestic animals varying in species, breed, age, and gender. Users create customer accounts that store their personal details and allow them to input funds to buy the animals on the program. The customers can filter the search for the pets they want based on their preferences. Staff members can update and monitor the system and update customer details.

Use Case Diagram



Specific Requirements

4.1 Revised Non-Functional Requirements

4.1.1 Operational Requirements

N1. The system has user-friendly menu-based navigation for both customers and staff.<Mandatory>

N2. The system has clear instructions for account creation, pet browsing, and adoption.<Mandatory>

N3. The system must ensure that pet details and account information are correctly stored and retrieved.<Mandatory>

4.1.2 Performance Requirements

N4. The system should handle up to 300 pets without noticeable performance degradation. <Mandatory>

N5. The Pet browsing and filtering should be instantaneous for users.

N6. The system should not crash during regular operations. <Mandatory>

N7. The system should handle invalid inputs (e.g., incorrect filters) and throw the right error message. <Mandatory>

N8. The system can be updated to include more advanced features like online access, notifications, etc. <Optional>

4.2 Revised Functional Requirements

4.2.1 Customer Features

F1. Customers can create new accounts. <Mandatory>

F2. Customers can log in to their accounts. <Mandatory>

F3. Customers can update their account password. <Mandatory>

F4. Customers can sort through pets using filters (type) <Mandatory>

F5. Customers can adopt pets and add them to their personal list. <Mandatory>

F6. Customers can view all the pets they have adopted. <Mandatory>

F7. Customers can deposit money into their wallets. <Mandatory>

4.2.2 Staff Features

F8. Staff members can create more new staff accounts. <Mandatory>

F9. Staff members can view or update customer account details. <Mandatory>

F10. Staff can view the total number of pets in the system. <Mandatory>

F11. Staff can update the total number of pets in the system. <Mandatory>

F12. Staff can view the total number of customer accounts. <Mandatory>

4.2.3 System Functional Requirements

F13. Pets are stored in a database and can be filtered by type. <Mandatory>

F14. The system tracks and stores customer and staff accounts. <Mandatory>

F15. The system enforces a maximum limit of 300 pets in its database. <Mandatory>

F16. The system prevents adding more than 300 pets. <Mandatory>

5. Justification

We initially had an idea for the program to have two modes: a Shelter Adoption Centre and a Specific Breed Store. The Shelter Adoption Centre would have a list or database of numerous pets varying in type, breed, age, gender, etc. Customers would be able to log in to their accounts and choose each mode they wanted. The Specific Breed Store would be able to calculate the price of specific breeds of animals that a customer chose and give links to various breeders with said breeds, whilst the Shelter Adoption Centre would just have random animals that customers could filter to their liking.

In the end, we decided to combine both modes, drop some of the features and keep the others. This was for ease of coding and also because we thought it would be too complicated to complete in the timeframe we had. We still kept the features of filtering and keeping a large amount of animals in the database. Each pet had a specific price and if a customer had enough money in his wallet on the program, they would be able to purchase the pet.

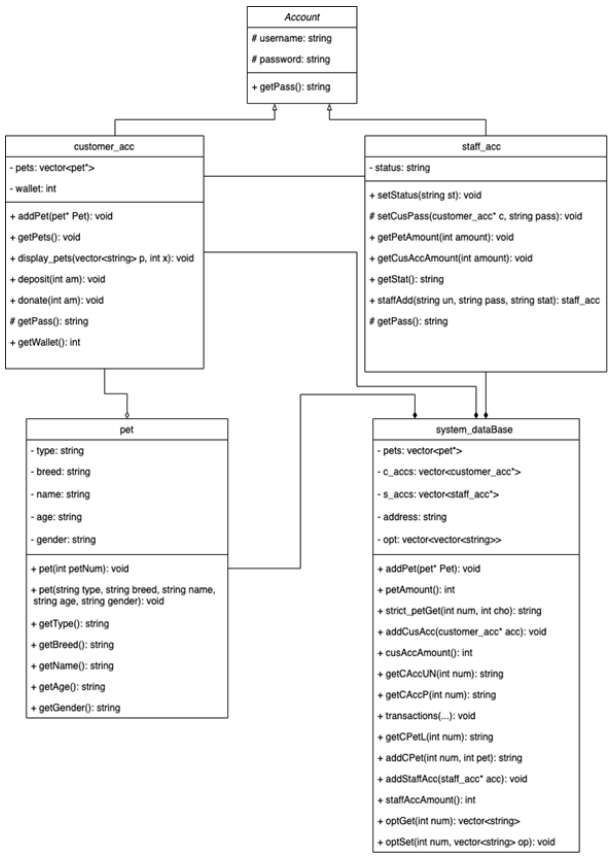
6. References

[1] IEEE Computer Society (1998). IEEE Std 830-1998: IEEE Recommended Practice for S SoftwareRequirementsSpecifications. https://standards.ieee.org/ieee/830/1222/

7. Introduction

This report provides a reflection of the design and requirements testing for our Adoption Pet Portal EE2010 Group 1 Project. It contains an evaluation of each of those phases and the experience gained through the project.

8. Class Diagram



Another UML diagram is not necessary, due to the relatively simple nature of the system's operation and the detail the class diagram already provides. The class diagram offers a clear overview of the system's structure including the classes, attributes and methods, which offers sufficient information about how the objects interact within the system. Key processes such as adding pets, handling transactions, or updating account balances are straightforward and can be inferred from defined methods like **'addPet'**, **'deposit'** and **'transactions'**. These methods, along with their interactions between objects like **'Customer_acc'**, **'Pet'**, and **'system_dataBase'**, do not require an additional dynamic representation to follow and understand. There are a limited number of method calls and object relationships that are not overly complex. The class diagram already shows how the objects interact, therefore adding another UML diagram would not significantly enhance the clarity or understanding of these operations.

9. Reflection on Design

The final design of our pet adoption system successfully met its primary objectives, delivering a modular and functional structure that facilitated the implementation of both customer and staff features. The use of distinct classes such as 'Account', 'Customer_acc', 'Staff_acc', 'Pet', and 'System_dataBase' were well-defined, with clear responsibilities and interactions. This separation of concerns helped the system maintain clarity, making the system architecture maintainable and scalable, and allowing us to implement features like filtering. The modularity of the design allowed each class to handle specific responsibilities, such as storing pet details or managing account operations. Overall, the design was effective in addressing the core requirements while remaining flexible enough for future extensions.

If we were to approach this project again, much of the existing design would remain intact. However, there are areas which could be enhanced. For instance, separating the filtering functionality into a dedicated class like 'FilteringUtility' would improve the structure, enabling more focused responsibilities and future developments would be easier to implement. Foundational principles of object-oriented programming, such as encapsulation and separation of concerns, is evident in the clear responsibilities of classes like 'Customer_acc', 'Staff_acc', and 'System_dataBase'. The 'System_dataBase' resembles a Singleton pattern by centralizing data management, though it is not strictly enforced. To further enhance

the design, formally implementing patterns like Singleton for System_dataBase, Factory for object creation, and Strategy for modular filtering could improve scalability, flexibility, and maintainability.

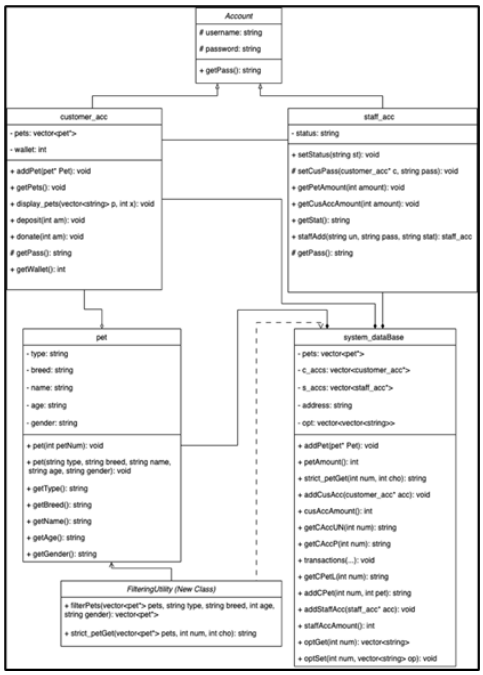


Figure 1: Illustration of 'FilteringUtility' class

The integration between classes was generally smooth, with clear relationships and interactions. For example, the 'System_dataBase' acted as the central repository, providing data to 'Customer_acc' and 'Staff_acc' for their specific operations. However, some methods, such as 'strict_petGet' in 'System_dataBase', felt overly specialized and might have been better refactored into a separate filtering utility class. While the overall integration was coherent, addressing these minor overlaps could further streamline the design.

By creating a new class called 'FilteringUtility', will allow the 'System_dataBase' to focus solely on its core responsibility of managing and storing data. Meanwhile, 'FilteringUtility' would handle the specialised filtering tasks, such as retrieving pets based on the customer's chosen criteria. This adjustment will improve the overall structure by promoting a cleaner separation of concerns, making the code more maintainable. Furthermore, it makes it easier to extend the filtering functionality in the future without affecting the core database operations and ensuring that each class is only responsible for its designated task.

The project presented a moderate level of difficulty, primarily due to the need to balance functionality and usability while adhering to both functional and non-functional requirements. Efficiently managing up to 300 pets and implementing error handling techniques, as required by Performance Requirement N7—ensuring the system handles invalid inputs (e.g., incorrect filters) and provides appropriate error messages—required thoughtful planning and thorough testing.

In conclusion, the design has met the project's goals effectively. The architecture is modular, and scalable, with each class fulfilling a clear responsibility. While there are a few areas for further improvement, such as the refactoring of specialized methods and the potential adoption of more advanced design patterns, the current core structure is solid. The integration between classes was clear, with some minor adjustments to reduce overlaps. Overall, the design led to a successful implementation product and the process was a valuable learning experience that will help guide future projects.

10. Reflection on Requirements Testing

- ***How effective was the Requirements testing at finding errors in your code?***

The requirements testing was effective in identifying errors in the code, particularly in edge cases where the system behavior differed from expected outcomes. For instance, during testing of the pet adoption feature, we discovered that the system did not properly validate input for selecting a pet when users attempted invalid selections. This allowed us to address specific logic flaws and improve input validation.

- ***Discuss whether you believe that your requirements test adequately tested the system?***

The requirements tests adequately covered the system by testing each core functionality, such as account creation, pet listing, adoption, and transaction handling. Using equivalence class partitioning, we ensured that a wide range of input types and scenarios were evaluated. However, some rare edge cases, such as simultaneous user interactions, were not fully tested due to time constraints. We believe that while the tests were comprehensive for core functionalities, additional stress and concurrency testing could enhance system reliability.

- ***Discuss whether you had to change the design as a result of requirements testing?***

The requirements testing process highlighted areas where design improvements were necessary to meet system requirements effectively.

Key modifications included:
Database Singularization:
It was observed that the system_database class could have been more focused and singular in responsibility. By separating functionalities, such as pet management and user account management, into distinct components, the design could have been more modular and maintainable.

Filter Functionality:
Issues were identified with the implementation of filters for pet listings. The complexity of handling various filtering criteria (e.g., type, breed, age, etc.) indicated the need for a dedicated Filter class. This would centralize the logic for filtering and make the system more extendable for future requirements.
Improved validation for account creation and password setting process

To achieve comprehensive testing:
Black-Box Testing: Used for modules like pet retrieval and account authentication, focusing on input-output behavior without delving into internal code.

White-Box Testing: Applied to critical functions such as addPet() and strict_petGet() to ensure all execution paths were covered.

- ***Did you have to create scenarios for the system to get it into a state to do the testing?***

Certain scenarios required preconditioning the system to specific states. For example:
Pre-loading the database with maximum pets (300) to test capacity constraints.
Creating mock user accounts to test authentication workflows.
Simulating invalid data entries to assess system robustness.

The reflection is organized by feature:

User Management: Tests validated the ability to create, authenticate, and manage user accounts, ensuring compliance with requirements.
Pet Management: Verifications included adding, retrieving, and listing pets

11. References

Source Making (2019). *Design Patterns and Refactoring*. [online] Sourcemaking.com-

Available at: https://sourcemaking.com/design patterns.

IEEE Standard (2008). *IEEE Standard for Software and System Test Documentation*. IEEE Std 829-2008. [online] Available at: https://standards.ieee.org/standard/829-2008.html

Unit Testing Report

12. Introduction

This document outlines the testing report for the system designed in the project. It covers the requirements testing plan and evidence of testing, detailing both functional and non-functional aspects of the system. The aim is to ensure that all requirements specified in the initial phases are met and validated through rigorous testing procedures. Screenshots and descriptions will be provided to confirm the correctness of the results.

13. Requirements Testing

This section documents the requirements testing for the system. It evaluates both functional and non-functional requirements to ensure compliance. For example, *functional requirements such as pet addition, customer account creation, and staff actions will be tested. Non-functional requirements like system stability will also be addressed*.

14.1 Requirements Test Plan

Explanation of this: see page 53 of IEEE Test Standard [2]. It provides a Level Test Case outline. The following is the example table I would suggest using. The row in grey are the guidelines and should not be included in your submission. Please note there may be more than one test per requirement to cover different possible input combinations. You need to include a table for all your requirements here

Test No.	Requirement No	Preconditions/ Dependencies	Expected Inputs	Expected Results	Test Evaluation

1	F1	Assume system is initialized successfully.	Input: Add a pet (Type: Dog, Breed: Husky, Name: Max, Age: 2, Gender: Male).	Display updated pet list with the newly added pet.	Validate output visually on the display.
2	F2	Assume system is initialized with accounts.	Input: Create customer account (Name: aymen, Password: 1205).	Confirm account creation and display account menu details in the system.	By review and by outputting the screen shot which confirms the output.
3	F3	System initialized and staff account available.	Input: Staff retrieves total number of pets and customer accounts.	Display total pets and accounts in the system accurately	Confirm through visual inspection.
4	F4	System initialized and pet database available.	Input: Filter pets by type = "Dog". (3)	Display list of all dogs stored in the system.	Validate output matches filter criteria.
5	F5	System initialized and pet/customer linked.	Input: Assign pet 'cat' to customer account "aymen".	Output should show pet assignment under customer account details.	Compare display with expected result.

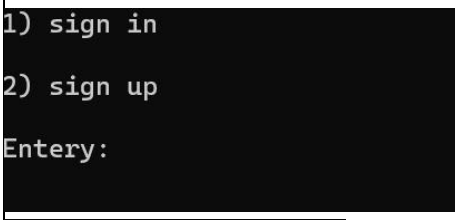
6	F6	System initialized with account and wallet setup.	Input: Deposit amount of £50 to "aymen"'s account.	Display updated wallet balance of "aymen" with the added amount.	Confirm wallet balance visually.

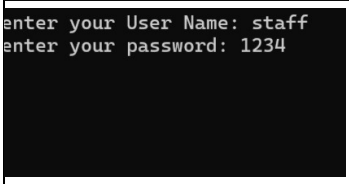
14.2 Evidence of the Testing

Test Number	Actual Output	Test Status

Successfully displayed "Dog, Husky, Max, Age 2, Passed Male".

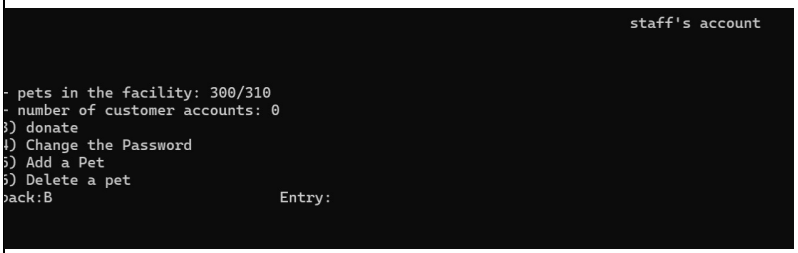
To select the account option in the menu, enter **2** when prompted. This will take you to the account menu where you can sign in or sign up.



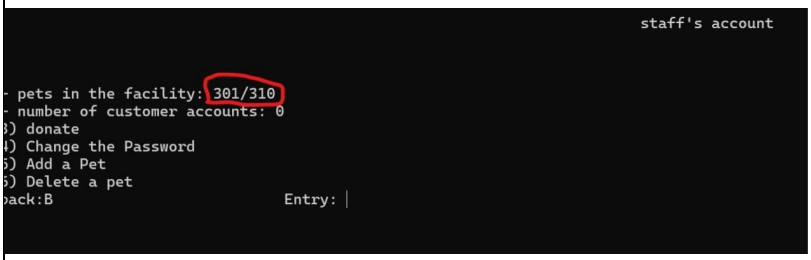


After selecting **2** for the account menu, you will be prompted to:

1. Enter your **username.**
2. Enter your **password.**

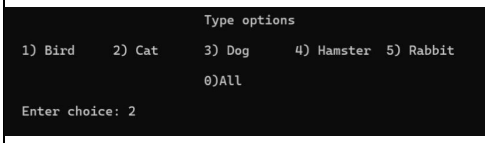
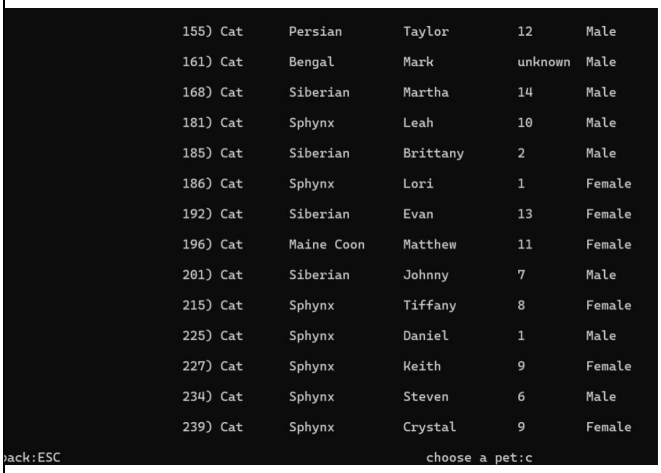
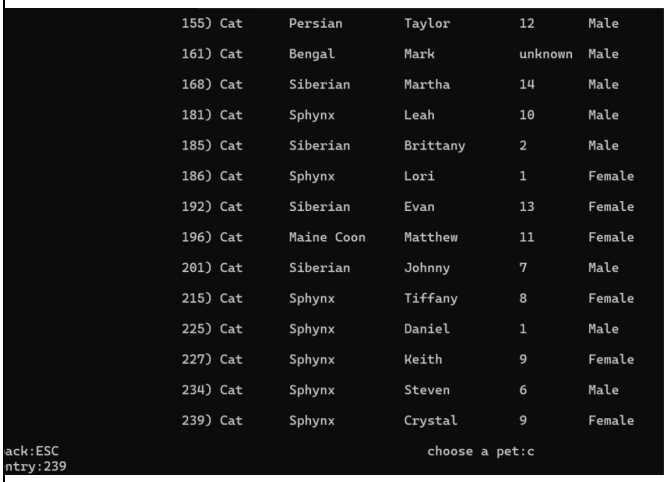
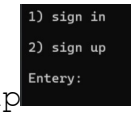
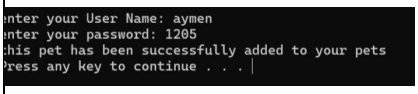



staff's account
- pets in the facility: 300/310
- number of customer accounts: 0
3) donate
4) Change the Password
5) Add a Pet
6) Delete a pet
back:B
Entry:
please enter the type of the pet: dog
enter breed: husky
enter name: max
enter age: 2
enter gender: male
!the pet has succesfuly been added!!Press any key to continue . . .


staff's account
- pets in the facility: 301/310
- number of customer accounts: 0
3) donate
4) Change the Password
5) Add a Pet
6) Delete a pet
back:B
Entry:


297) Dog Poodle Justin 8 other
298) Dog Poodle Stephanie 6 other
299) Cat Sphynx Jason 7 other
300) Rabbit Angora Michelle 9 other
301) dog husky max 2 male

2	Customer account: aymen, Password: 1205" displayed correctly.) sign in) sign up ntery: SIGN IN nter your User Name: aymen nter your password: 1205 our account has been createdPress any key to continue . . . aymen's account wallet: 0) My Pets) Deposit) donate) Change the Password ack:B Entry:	Passed
3	Displayed "Total pets: 301, Total accounts: 2" accurately. staff's account pets in the facility: 301/310 number of customer accounts: 2) donate) Change the Password) Add a Pet) Delete a pet ack:B Entry:	Passed
4	Correctly filtered and displayed all "Dog" type pets. Type options 1) Bird 2) Cat 3) Dog 4) Hamster 5) Rabbit 0)All Enter choice: 3 type breed name age gender 5) Dog Labrador Adam 2 Male 7) Dog Beagle Marcus 3 Male 9) Dog Labrador Katherine 1 Female 12) Dog Bulldog David unknown Male 14) Dog Bulldog Alan 10 other 15) Dog Labrador Michael 10 Female 16) Dog Labrador Susan 6 Female 18) Dog Beagle Corey unknown Male 38) Dog Poodle Anna 14 Female 41) Dog Labrador Sean 13 Female 42) Dog German ShepherdBrittany 13 Female 70) Dog German ShepherdBrian 8 Male 73) Dog Bulldog Isaac 8 Female 78) Dog German ShepherdAnna 11 Male 79) Dog Labrador Donald 13 Male ack:ESC choose a pet:c	Passed

5	Pet ‘cat’ successfully assigned to "aymen".   Write 2 to select the cat option    sign up  	Passed

```
155) Cat      Persian      Taylor       12       Male
161) Cat      Bengal       Mark         unknown  Male
168) Cat      Siberian     Martha       14       Male
181) Cat      Sphynx       Leah         10       Male
185) Cat      Siberian     Brittany     2        Male
186) Cat      Sphynx       Lori         1        Female
192) Cat      Siberian     Evan         13       Female
196) Cat      Maine Coon   Matthew      11       Female
201) Cat      Siberian     Johnny       7        Male
215) Cat      Sphynx       Tiffany      8        Female
225) Cat      Sphynx       Daniel       1        Male
227) Cat      Sphynx       Keith        9        Female
234) Cat      Sphynx       Steven       6        Male
239) Cat      Sphynx       Crystal      9        Female
ack:ESC                                   choose a pet:c
```

```
                                   Welcome                                        *
                                      to                                          *
                              Pet Adoption Center                                 *
                                                                                  *
**********************************************************************************

                              1) Adopt a Pet
                              2) account

ntry: 2
```

```
1) sign in
2) sign up
Entery:
```

Enter your username and password to log in. Ensure the credentials are correct to proceed successfully.

```
nter your User Name: aymen
nter your password: 1205
```

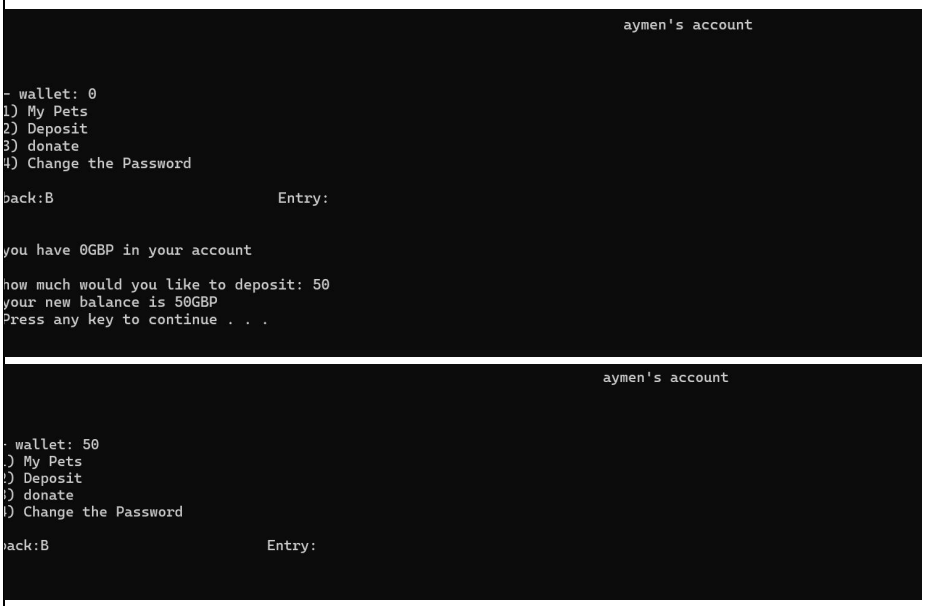
```
                                                   aymen's account

 wallet: 0
) My Pets
) Deposit
) donate
) Change the Password

ack:B                          Entry:

                        type     breed          name        age      gender
                      1) Cat     Sphynx         Crystal     9        Female

ress any key to continue . . .
```

5	Wallet balance: £50" correctly updated.  	Passed

15. References

[2] IEEE Computer Society (1998). IEEE Std 830-1998: IEEE Recommended Practice for SoftwareRequirementsSpecifications. https://standards.ieee.org/ieee/830/1222/