

Design Document
for
Web Cattery Management System

Version 1.0 approved

Prepared by: Sherry Zhao, Ningjia Huang, Zora Li, Nora Tang, Zhaokuan Chen

Case Western Reserve University

Apr 2, 2021

1. Introduction	4
1.1 Purpose	4
1.2 Design Patterns	5
2. Principal Classes	5
2.1 Account	5
2.2 TableAccess	6
2.3 CatInfo	6
2.3.1 PregnantCatInfo	6
2.4 GeneticCalculator	7
2.5 FamilyTree	7
3. Interactions between components	7
3.1 Interaction between Admin and TableAccess	7
3.1.1 Description	7
3.1.2 Sequence Diagram for Admin interacting with TableAccess	9
3.2 Interaction between Account and TableAccess	9
3.2.1 Description	9
3.2.2 Sequence Diagram for Account interacting with TableAccess	10
3.3 Interaction between CatInfo and TableAccess	11
3.3.1 Description	11
3.3.2 Sequence Diagram for updating/searching CatInfo data	13
3.4 Interaction between GeneticCalculator, CatInfo, and TableAccess	14
3.4.1 Description	14
3.4.2 Sequence Diagram for calculating genetic results	15
3.5 Interaction between FamilyTree and TableAccess	15
3.5.1 Description	15
3.5.2 Sequence Diagram for generating family tree of a cat	16
4. Class Interface	17
4.1 TableAccess	17
4.1.1 void insert(String1, String2, String3, ...)	17
4.1.2 List<String> search(String1, String2, String3, ...)	17
4.1.3 void update(String1, String2, String3, ...)	17
4.2 Account	17
4.2.1 void login(String username, String password)	17
4.2.2 void signUp(String username, String password, String accountType)	17
4.2.3 boolean validate(String username, String password)	17
4.2.4 void forgetPassword(String username)	18
4.2.5 boolean logout()	18
4.3 Admin	18
4.3.1 void createAccount(String username, String password)	18
4.3.2 boolean deleteAccount(String username)	18

4.3.3 void editAccount(args[])	18
4.3.4 void readAccounts()	18
4.4 Breeder	18
4.4.1 string[] search(String catName)	18
4.5 PotentialParents	19
4.5.1 string[] search(string catName)	19
4.8 GeneticCalculator	19
4.8.1 List<Double> getGeneInfo(String name)	19
4.8.2 List<Double> getGeneInfo(int certificationNO)	19
4.8.3 List<Double> calculate(List<Double> data1, List<Double> data2)	19
4.9 FamilyTree	20
4.9.1 List<String> search(String name)	20
4.9.2 List<String> search(int certificateNO)	20
4.9.3 void pair(String newCat, String family)	20
4.10 CatInfo	20
4.10.1 List<String> search(args[])	20
4.10.2 List<String> save(agsr[])	20
4.10.3 List<String> patchUp(String newCat, String family)	20
4.10.4 boolean validate(args[])	20
4.10.5 List<String> pair(args[])	20
4.10.6 void close()	20
Appendix A: Database ER diagram design	21
Appendix B: UI Design	22
B.1 Login Page:	22
B.2 Gene Calculator:	22
B.3 CatInfo:	22
B.4 PregnantCatInfo:	23
B.5 Family Tree:	23
Appendix C: References	24

1. Introduction

As written in our software requirements specification, the purpose of the Web Cattery Management System is to collect kittens and cats' information in different catteries and be able to find catteries registered under CFA, TICA, and FIFe [1]. Not only will our website contain details about cat mating and pregnancy, there is also a Family tree function and a gene calculator available [1]. The Web Cattery Management System is expected to evolve as the information about cats and kittens changes [1].

1.1 Purpose

To smoothly develop our Web Cattery Management System and achieve the goal of this project in a timely manner, we made this design document to describe the details of the design of our website. Different sections are indicated as following in Table 1:

Table 1. Different sections of the design document

Section	Content
Section 2	Identify and define the five principal classes in our project
Section 3	Illustrate the five main interactions between components and classes
Section 4	Design the interface of the website's main functions

According to Table 1, section 2 explains all principal classes used in our website in detail. The principal classes include Account, TableAccess, CatInfo, GeneticCalculator, and FamilyTree. As illustrated in Figure 1, these principal classes can be divided into two levels according to their functions, the level of front end and the level of back end. The classes that have more interactions with the user interface are classified as the classes of front-end level. For example, the FamilyTree class displays the pedigree of a certain cat, including his predecessors, siblings, and offspring if there is any. In contrast, the classes that do not interact with user interfaces will be classified as the classes of back-end level. For example, the Account class mostly contains functions that interact with the database.

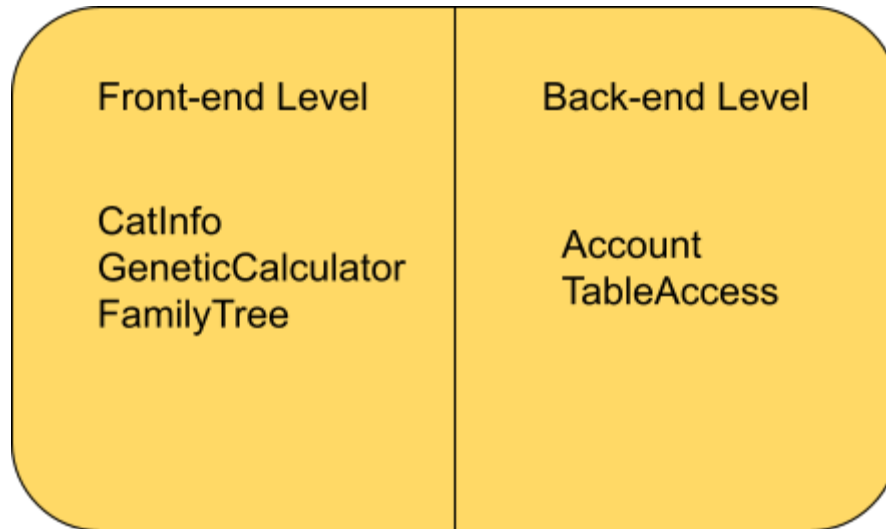


Figure 1. Front-end principal classes and back-end principal classes

In section 3, five main interactions between components and classes are described, including interaction between Admin and TableAccess, interaction between Account and TableAccess, interaction between TableAccess and CatInfo, interaction among GeneticCalculator, CatInfo, and TableAccess, and interaction between FamilyTree and TableAccess. Each interaction between components is demonstrated by a comprehensive description and a sequence diagram. The TableAccess class supports these interactions.

The main functions of our website are explained in section 4, including insert, search, validate, and so forth. Their functions with front and back end are achieved through interactions with other classes after handling their own parameters.

1.2 Design Patterns

This project is going to implement observer pattern design. The observer will be the catInfo class. Therefore, when an object changes its state, all of the dependents will be notified and automatically get updated.

2. Principal Classes

This section defines the main classes we are going to implement in this project. The bracket after each functionality specifies the corresponding functional requirements in the Software Requirement Specification.

2.1 Account

The Account class is responsible for the following functionalities:

- Breeder/Parent/Administrator login. [LI-F1, LI-F4-LI-F8]

- Create a new breeder/parent/administrator account. [LI-F1-LI-F3, LI-F9-LI-F13]
- In the breeder login interface, if the information does not match with any account in the database, the system saves the information to a query box and waits for administrator's approval. [LI-F13]
- Forgot password feature. [LI-F14]

All the breeders, parents, and administrators will have an account to access the system. The Account class will collect users' information from the user input and handle login information as well as user information stored in the login database through TableAccess class. The Account class will have three subclasses: BreederAccount, ParentAccount, and AdminAccount. The interface of creating new accounts and login should be identical while their accessibility to functionality from other classes are different.

2.2 TableAccess

Since all of the classes require TableAccess class to query, insert, update, and delete data, we will specify the functional requirements in those classes instead of the helper class TableAccess. The Table Access class is responsible for the following functionalities:

- Convert reading/writing functionalities.
- Read data from the database.
- Insert data into the database.
- Modify data in the database.
- Delete data from the database.

To some extent, all of the classes will make use of the TableAccess class to handle the interactions with the databases. The TableAccess class has the accessibility to the UserInfo table and the CatInfo table to handle login/creating new account requests and cat information requests.

2.3 CatInfo

The CatInfo class is responsible for the following functionalities:

- Register new cat information. [CI-F1-CI-F4]
- Query cat information by providing name of the cat or the certificate number of the cat. [UC-F1-UC-F2]

The CatInfo class mainly handles the query request of cat information. It will interact with the TableAccess class and use the functions to query or update information stored in the CatInfo table. Based on the different requirements on fields and features, the class PregnantCatInfo will inherit the CatInfo class.

2.3.1 PregnantCatInfo

The PregnantCatInfo class will have the following extra features:

- Add new weights and generate a weight curve for the pregnant cat. [PM-F2-PM-F4]
- Add new health examination records. [PM-F2]

The pregnant cats will have the same features as other cats, except they will have additional features such as daily weights and more health examination records. Pregnant cats can be identified by a String attribute in the CatInfo table. As the PregnantCatInfo class makes use of the TableAccess class, it will filter out the cats with that column as “not pregnant”.

2.4 GeneticCalculator

The GeneticCalculator class is responsible for the following functionalities:

- Allow the breeder to enter the name of sire and dam into textboxes. [GC-F1]
- Query the disease information of the sire and the dam. [GC-F2]
- Perform calculations on the potential diseases the kitten may suffer and generate a report. [GC-F2]

The GeneticCalculator class makes use of the CatInfo class to query the disease history of the sire and dam through CatInfo Class and calculate the potential diseases based on the gene protocol.

2.5 FamilyTree

The FamilyTree class is responsible for the following functionalities:

- Allow the user to input the name or certificate number of a cat. [FT-F1]
- Generate the family tree of the cat that the user is querying. [FT-F2]
- Allow the breeder to pair a cat with another cat to form a family tree. [FT-F2]

The FamilyTree class is used to specify the parents-child relationship between the cats through CatInfo class and visualize the query results as a family tree.

3. Interactions between components

3.1 Interaction between Admin and TableAccess

3.1.1 Description

In the interaction between Admin and TableAccess, we use TableAccess class to help Admin class modify the User table in the database. TableAccess will translate the order given by the administrator into SQL language to perform corresponding functions.

Option 1: Admin creates an account

Admin calls createAccount(username, password) request to TableAccess. TableAccess receives the account that needs to be created and calls insert() to translate it into the format: INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...). Then, the database receives and runs the command.

Option 2: Admin deletes account

Admin calls deleteAccount(username) request to TableAccess. TableAccess receives the account name that needs to be deleted and calls delete() to translate it into the format: DELETE FROM table_name WHERE condition. Then, the translated SQL command is sent to the database and the command is run by the database system. The TableAccess.delete(username) will return a boolean which will trigger a system alert to alert the Admin if the delete is not successful.

Option 3: Admin reads all accounts

Admin sends readAccounts() requests to TableAccess. TableAccess receives the call and calls get() to translate it to SQL format: SELECT * FROM table_name. All accounts' information will be selected and returned to the Admin.

Option 4: Admin updates information of an account

Admin sends editAccount(account) requests to TableAccess. TableAccess receives the new version of the account information and calls edit(account) to translate it into SQL format: UPDATE table_name SET col1 =val1, col2 = val2, WHERE condition. Then the corresponding account will be updated in the database.

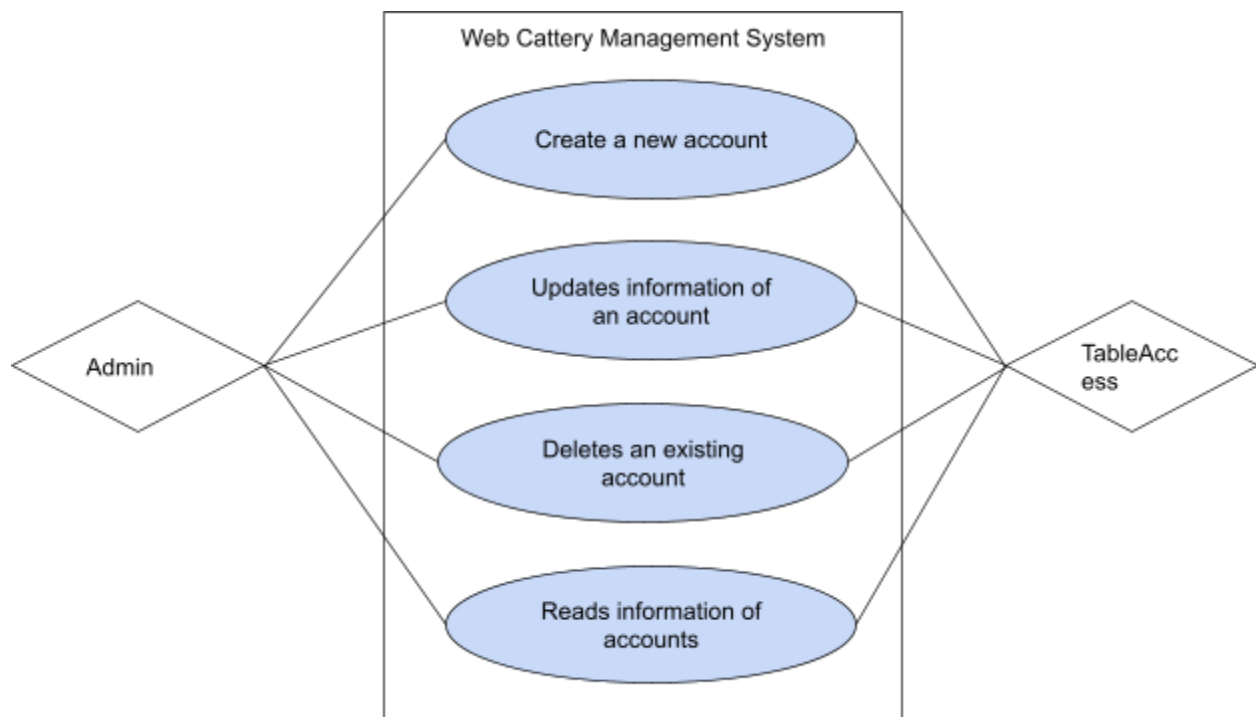


Figure 2: An example explains the interactions between Admin and TableAccess

3.1.2 Sequence Diagram for Admin interacting with TableAccess

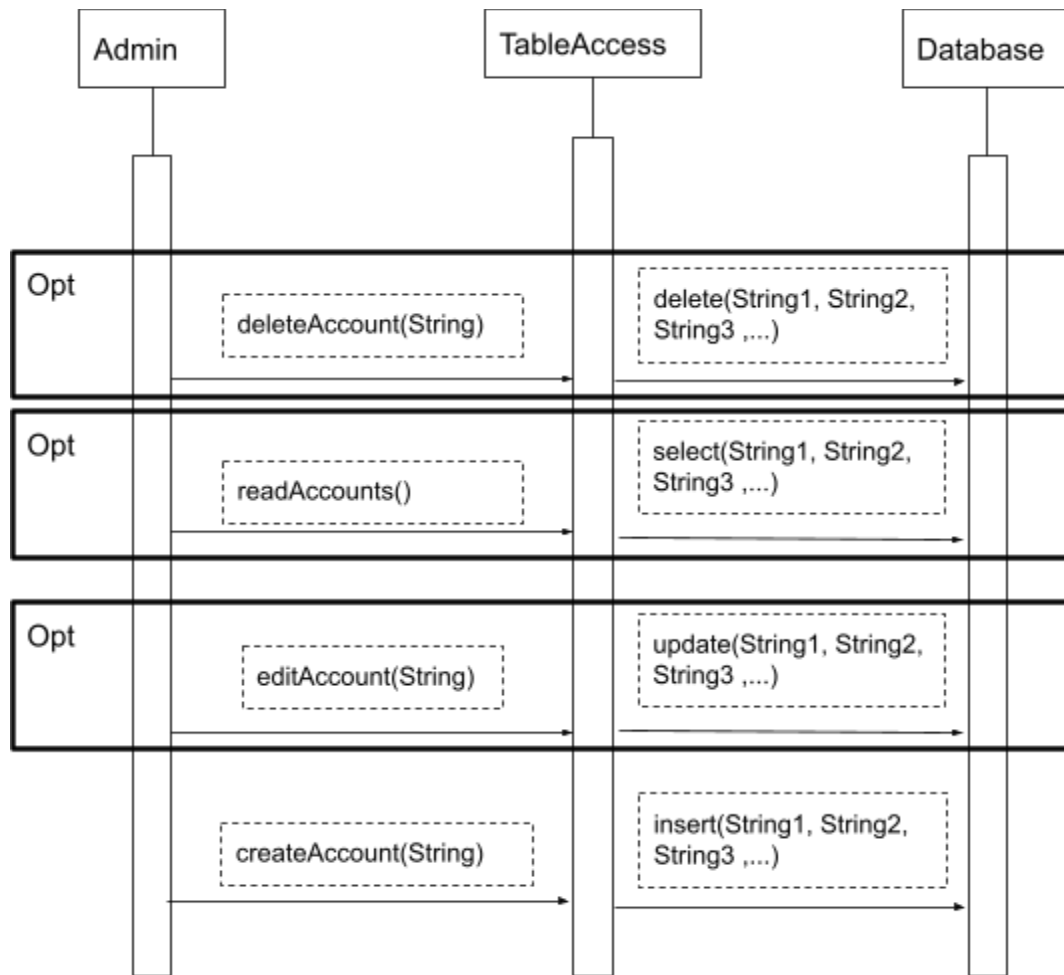


Figure 3: Sequence diagram for interactions between Admin and TableAccess

3.2 Interaction between Account and TableAccess

3.2.1 Description

In the interaction between User and TableAccess, we use TableAccess class to help Account class interact with the system. TableAccess will translate the order given by the administrator into SQL language to perform corresponding functions.

Option 1: User logs into the account

First, the Account class will call the login(String username, String password) method. Then it follows by triggering validate(String username, String password) to check if the input information is correct. If not, validate(String username, String password) method will send an alert message to the user to modify that textbox until it is correct.

Option 2: User signs up an account

Account sends signUp(String username, String password, String accountType) request to TableAccess. It will trigger select(args[], string) which translates the given command into SQL language in the form: SELECT given_acct_info FROM table_name. If not, signUp method will trigger insert(args[], string) to create a new account in the database. If it does exist, the system will send a message to the user.

Option 3: User forgets password of the account

Account sends forgetPassword(String username) request to TableAccess. It will trigger select(args[], string) to check if the username exists in the database. The command is translated into SQL language in the form: SELECT given_acct_info FROM table_name. If not, the system will send an alert message to the user. If it does exist, it will trigger validate(args[]) to validate the identity of the user. If the user passes the validation, the system will send the account information to the user.

Option 4: User logs out the account

Account calls logout() request. The system will send a message to the user to indicate if the user successfully logs out the account.

3.2.2 Sequence Diagram for Account interacting with TableAccess

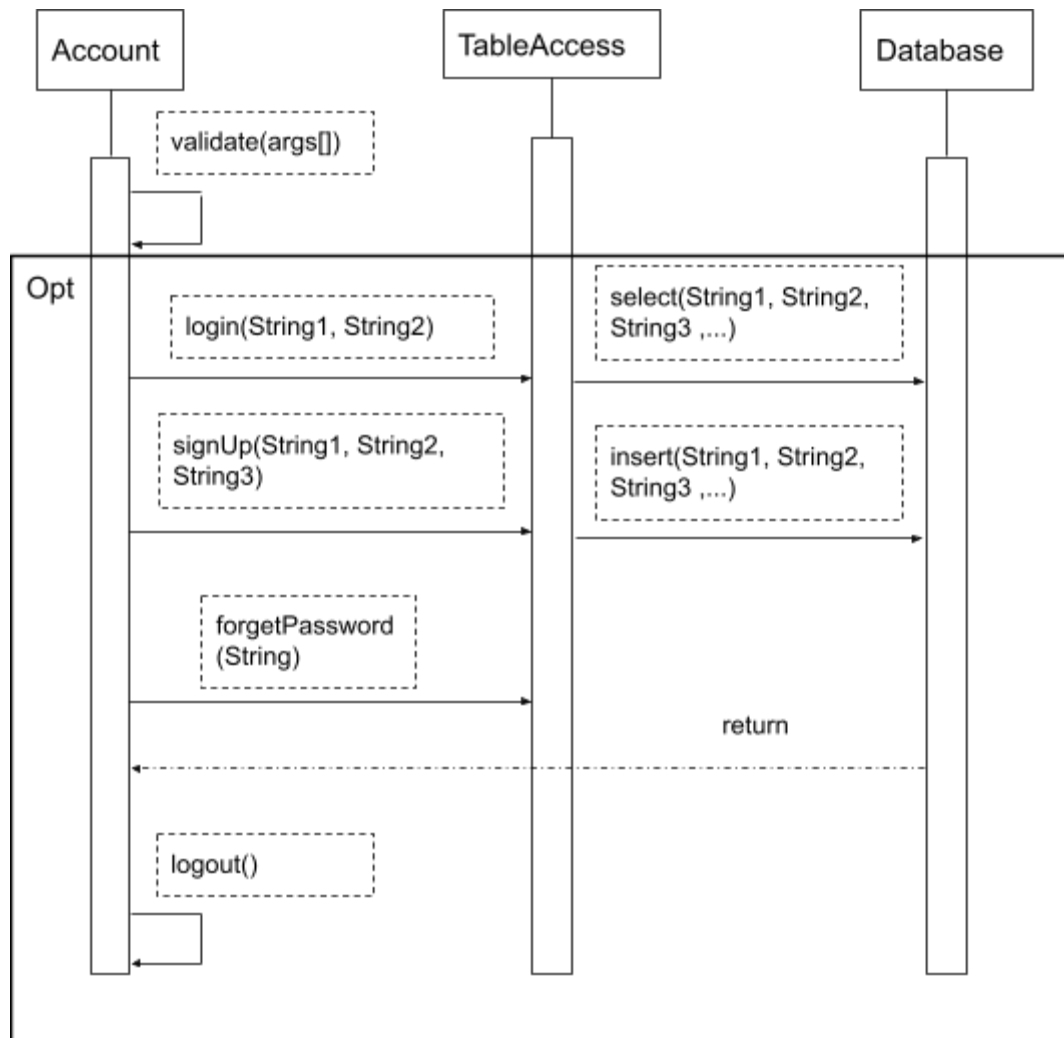


Figure 4: Sequence diagram for interactions between Account and TableAccess

3.3 Interaction between CatInfo and TableAccess

3.3.1 Description

In the interaction between TableAccess and CatInfo, TableAccess class is designed to translate the order given by CatInfo class to update or retrieve requested information in the database. The command is in the format: SELECT given_info FROM table_name or INSERT INTO table_name(col1, col2, ...) VALUES (val1, val2, ...).

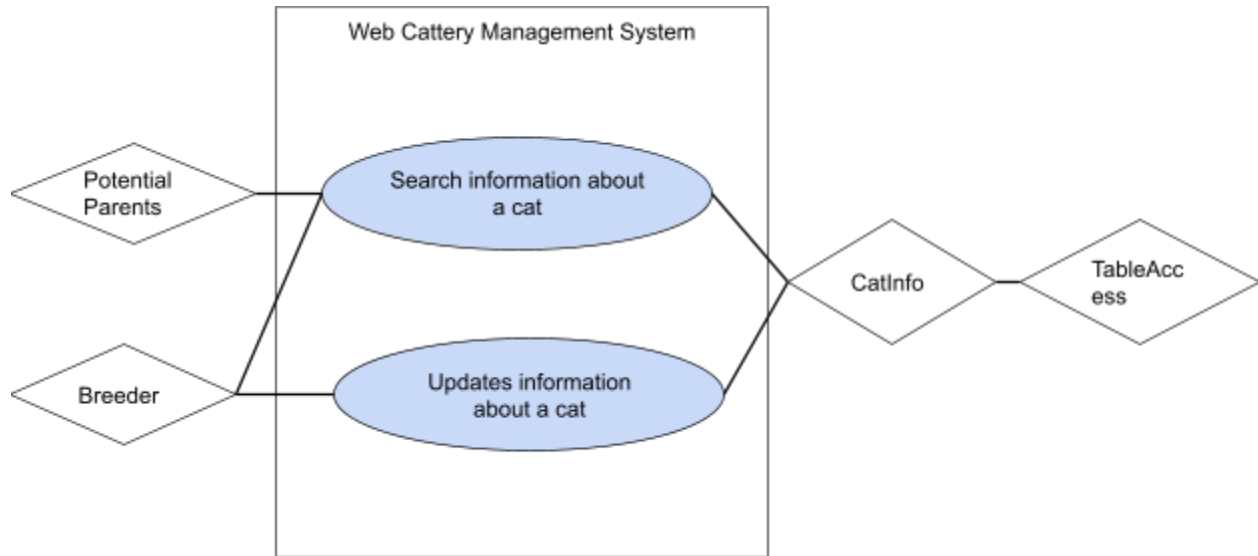


Figure __: An example explains the interactions between CatInfo and TableAccess

Option 1: Breeders or Potential Parents search information of a cat

First, for the search action, CatInfo will call the validate() method to check if any textbox is null or answered in an unacceptable format. If an unacceptable format is input, validate() method will send an alert message to the user to modify that textbox. Until the user inputs a correct format, CatInfo will call the search(args[]) where the array is the user's input. Then, search(args[]) will trigger the select(args[], string) which translates the given command into SQL language in the form: SELECT given_info FROM table_name.

Option 2: Breeders inserts information of a cat

For the update action, CatInfo will first do the validation process as described above. Until the user inputs a correct format, CatInfo will call the save(args[]) where the array is the user's input. Then, save(args[]) will trigger the insert(args[], string) which translates the given command into SQL language in the form: INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...).

Option 3: Breeders updates information of a cat

For the update action, CatInfo will first do the validation process as described above. Until the user inputs a correct format, CatInfo will call the patchUp(args[]) where the array is the user's input. Then, patchUp(args[]) will trigger the update(args[], string) which translates the given command into SQL language in the form: UPDATE table_name SET col1 =val1, col2 = val2, WHERE condition.

Option 3: Breeders pair/mate two cats

For mating action, CatInfo will first do the validation process as described above. Until the user inputs the correct format, CatInfo will call the pair(String king_name, String queen_name) to generate a SQL command that is sent to the TableAccess class. The pair method will first trigger a search method to check if the input cat names are in the database. If not, the system will send an alert. If it does exist, it will trigger the update method to update the pairing record of two cats. The SQL command is in the form: UPDATE table_name SET col1 =val1, col2 = val2, WHERE condition.

In addition, the user can choose to close the window by triggering the close button which is embedded with close() method at any time. Any unsaved variable will be deleted and cleared.

3.3.2 Sequence Diagram for updating/searching CatInfo data

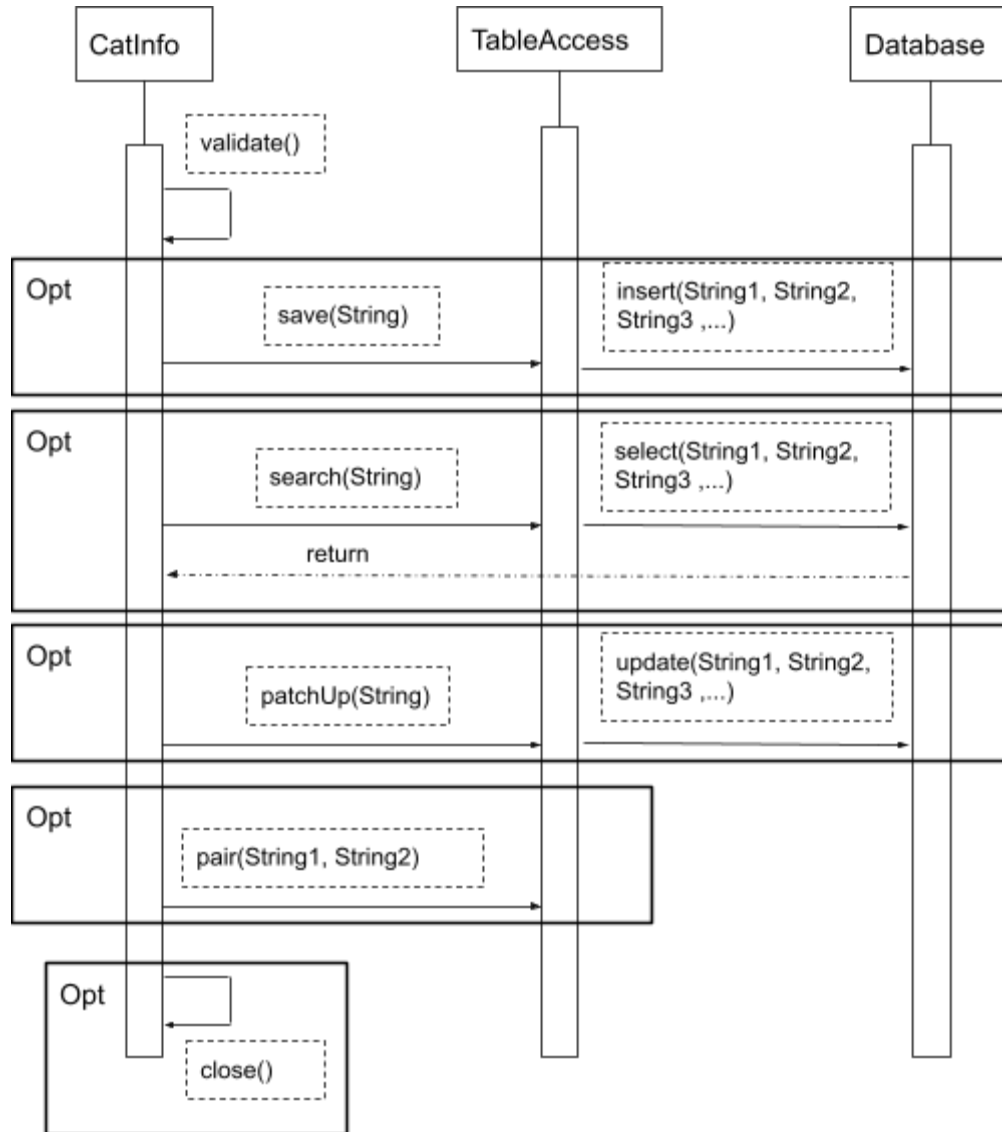


Figure 5: Sequence diagram for interactions between CatInfo and TableAccess

3.4 Interaction between GeneticCalculator, CatInfo, and TableAccess

3.4.1 Description

In the interaction between GeneticCalculator, CatInfo, and TableAccess, GeneticCalculator class is designed to calculate the possibility of genetic diseases of the child of two cats given the information from CatInfo. CatInfo will be given the command to generate inputs for GeneticCalculator in the format: `SELECT given_info FROM table_name`.

To get the information from a certain cat, the GeneticCalculator will first call `getGeneInfo(String)` or `getGeneInfo(int)` to retrieve the genetic information with the input of the

cat's name or certification number. It follows by triggering the validate() method. If it passes, CatInfo will call the search(args[]) where the array is the user's input. Then, search(args[]) will trigger the select(args[], string) which translates the given command into SQL language in the form: SELECT given_info FROM table_name.

Then, with the retrieved information, GeneticCalculator class then calls calculate(args[]) to calculate the possible genetic diseases for the cat.

In addition, the user can choose to close the window by triggering the close button which is embedded with close() method at any time. Any unsaved variable will be deleted and cleared.

3.4.2 Sequence Diagram for calculating genetic results

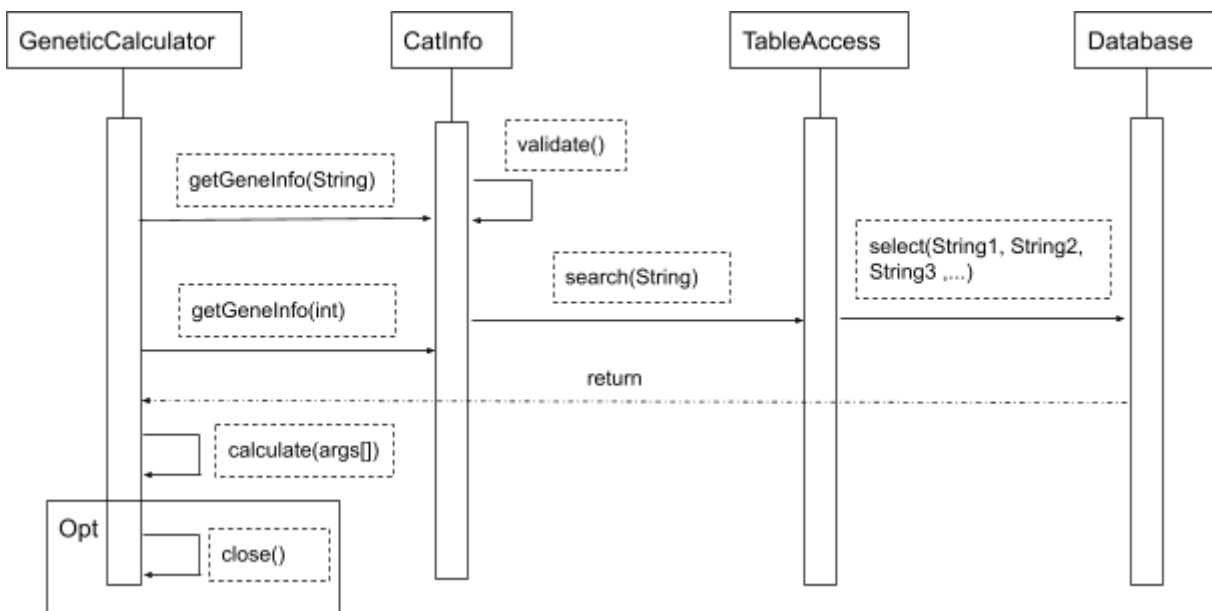


Figure 6: Sequence diagram for GeneticCalculator to get the result

3.5 Interaction between FamilyTree and TableAccess

3.5.1 Description

In the interaction between FamilyTree and TableAccess, FamilyTree class is designed to search and display the family tree of a cat. TableAccess will be given the command to help retrieve the family tree of the cat. The command in the format: SELECT given_info FROM table_name.

Option 1: Breeders or Potential Parents search family tree of a cat

The FamilyTree class will call search(args[]) with inputs of the user (either cat name or certification number). Then, search(args[]) will trigger the select(args[], string) which translates the given command into SQL language in the form: SELECT sire, dam FROM cat_name.

Option 2: Breeders pair the family members of a cat

The FamilyTree class will call pair(args[]) with inputs of the user (either cat name or certification number). Then, pair(args[]) will trigger the update(args[], string) which translates the given command into SQL language in the form: UPDATE table_name SET col1 =val1, col2 = val2, WHERE condition.

3.5.2 Sequence Diagram for generating family tree of a cat

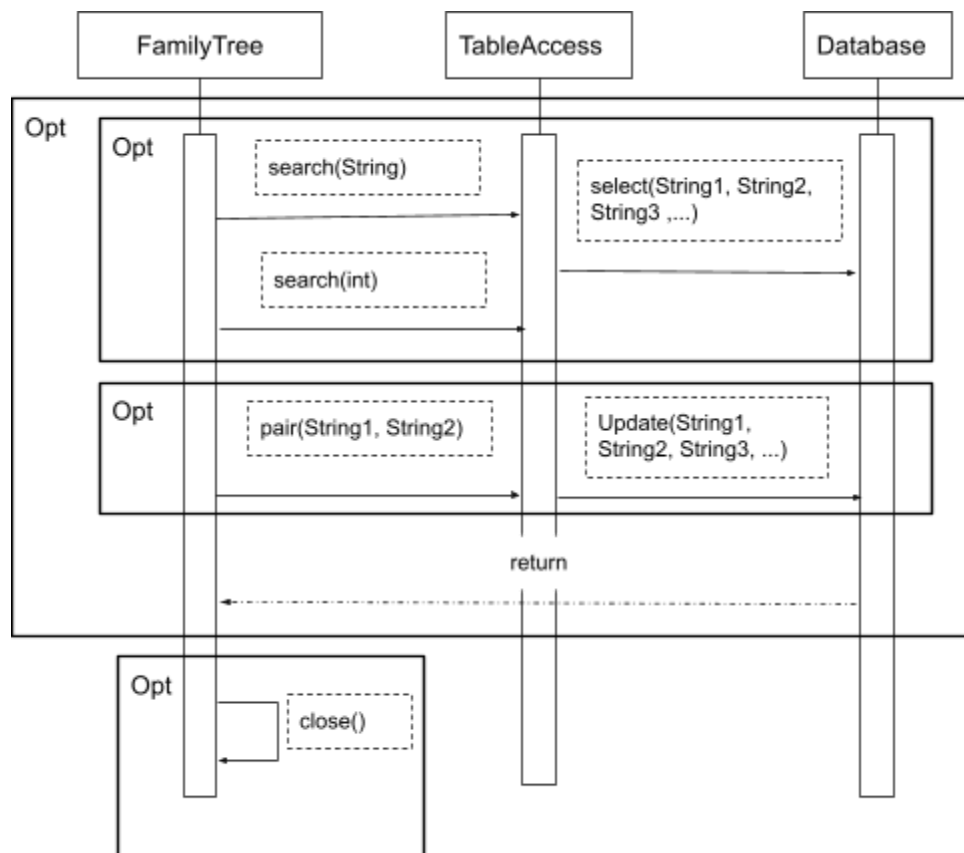


Figure 7: Sequence diagram to get the family tree of a cat

4. Class Interface

4.1 TableAccess

An instance of class connects the database with other classes. It contains methods that allow other classes to interact with the database.

4.1.1 void insert(String1, String2, String3, ...)

The method insert() will insert new information to the database. String1, String2, String3... are the string data we want to insert into the database.

4.1.2 List<String> search(String1, String2, String3, ...)

The method search will return a list of string data required by the input. String1, String2, String3... are the input we use to locate the required data.

4.1.3 void update(String1, String2, String3, ...)

The method update() will update new information in the database. String1, String2, String3... are the string data we want to update into the database.

4.2 Account

An instance of class represents the user of the software. It contains the basic methods that can be used by the users.

4.2.1 void login(String username, String password)

The method will let the users log into their account with their username and password. The username in the method represents the username of the user's account, and the password in the method represents the password of the user's account.

4.2.2 void signUp(String username, String password, String accountType)

The method will let the users log sign up for an account with their username and password. The username in the method represents the username of the user's account, and the password in the method represents the password of the user's account. The input accountType is the identity of the user (Administrator, breeder, or potential parent).

4.2.3 boolean validate(String username, String password)

The method will validate if the users can log into their account. The username in the method represents the username of the user's account, and the password in the method represents the password of the user's account. If the user forgets the password, the system validates the identity of the user with other information. The method returns if the user's identity is validated.

4.2.4 void forgetPassword(String username)

The method will let the user reset the password after validating the user's identity. The username in the method represents the username of the user's account.

4.2.5 boolean logout()

The method will log out the user account. The method will return if the account is logged out successfully.

4.3 Admin

4.3.1 void createAccount(String username, String password)

The method will create a new account in the database. The input username represents the **username** of the new account, and the input password represents the password of the new account.

4.3.2 boolean deleteAccount(String username)

The method will delete an account in the database. The input username represents the username of the account. The method will return if the account is deleted successfully.

4.3.3 void editAccount(args[])

The method will update the information of the specified account. The input is the information needed to be updated.

4.3.4 void readAccounts()

The method will return all the accounts in the database.

4.4 Breeder

4.4.1 string[] search(String catName)

The method will search for the cat in the database. The input catName represents the name of the cat. The method will return an array of type string. The array will contain all the information of a certain cat the user is searching for.

4.4.2 void pair(String cat1, String cat2)

The method will pair two cats in the database. The input cat1 and cat2 represent the first cat and the second cat respectively. The method will return nothing.

4.4.3 void update(args[])

The method will update information of the specified cat. The input is the information needed to be updated.

*search(), and pair() methods will be passed to CatInfo class.

4.5 PotentialParents

4.5.1 string[] search(string catName)

The method will search for the cat in the database. The input catName represents the name of the cat. The method will return an array of type string. The array will contain all the information of a certain cat the user is searching for.

*search() method will be passed to CatInfo class.

4.8 GeneticCalculator

4.8.1 List<Double> getGeneInfo(String name)

The method gets the genetic information of a cat from the database. The input name represents the name of a cat. The method returns a list of data representing the genetic information of the cat.

4.8.2 List<Double> getGeneInfo(int certificationNO)

The method gets the genetic information of a cat from the database. The input certificationNO represents the certification number of a cat. The method returns a list of data representing the genetic information of the cat.

4.8.3 List<Double> calculate(List<Double> data1, List<Double> data2)

The method calculates the possibility of genetic diseases of the child of two cats. The input data1 represents the genetic diseases of one cat, and the input data2 represents the genetic diseases of the other cat. The method returns the possibility of genetic diseases of the child.

4.9 FamilyTree

4.9.1 List<String> search(String name)

The method will search the family tree of a cat. The input name represents the name of the cat we want to search. The method returns the family tree of the cat.

4.9.2 List<String> search(int certificateNO)

The method will search the family tree of a cat. The input name represents the certification number of the cat we want to search. The method returns the family tree of the cat.

4.9.3 void pair(String newCat, String family)

The method will put a cat into the family tree and pair the cat with its family members. The input newCat represents the name of the new cat inputted into the family tree, and the family is the name of the existing cat in the family tree that the new cat is paired with.

4.10 CatInfo

4.10.1 List<String> search(args[])

The method will generate the SQL command for searching action in the database.

4.10.2 List<String> save(ags[])

The method will generate the SQL command for saving action in the database.

4.10.3 List<String> patchUp(String newCat, String family)

The method will generate the SQL command for updating action in the database.

4.10.4 boolean validate(args[])

The method will validate if the given input is in the correct format.

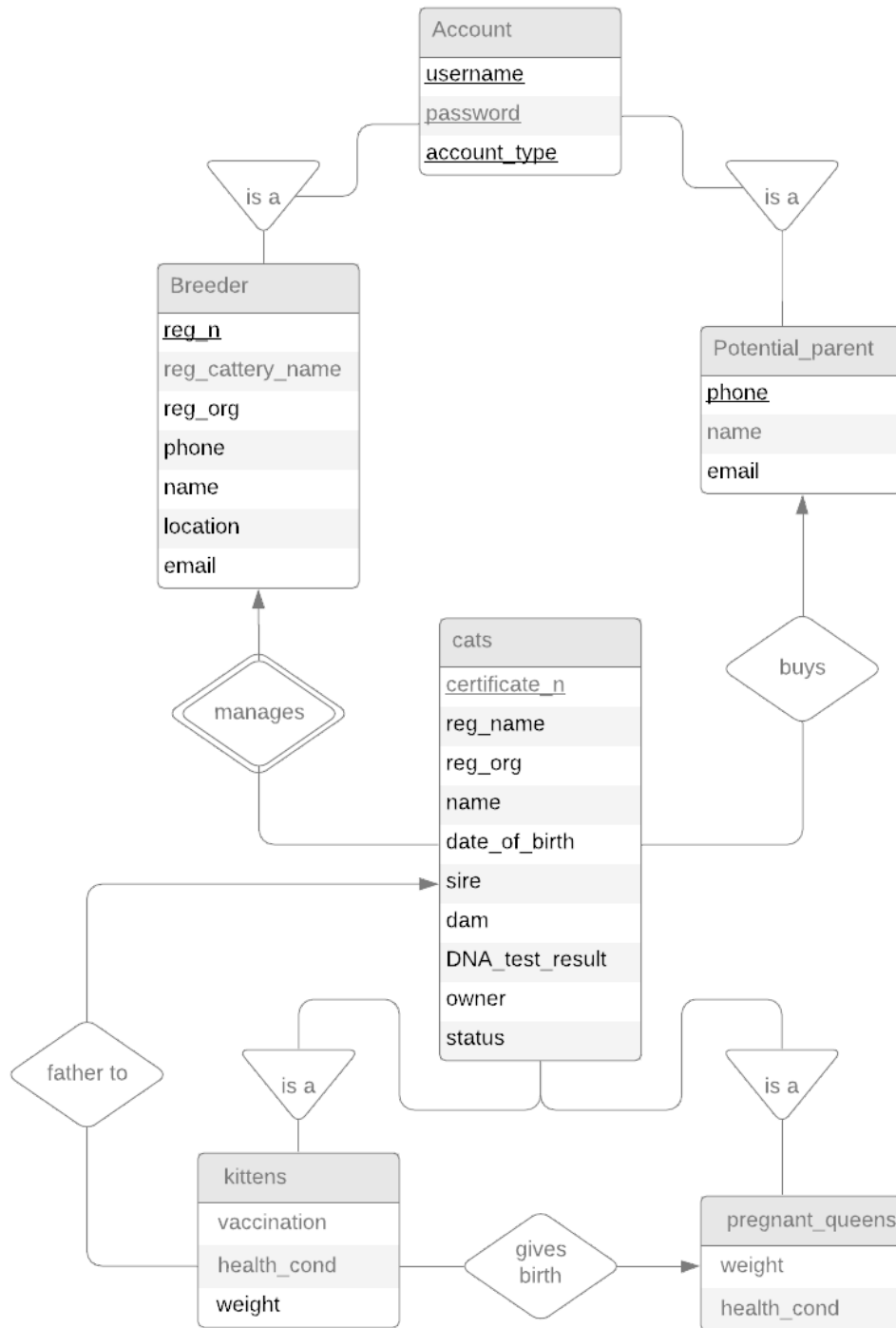
4.10.5 List<String> pair(args[])

The method will generate the SQL command for pairing action in the database.

4.10.6 void close()

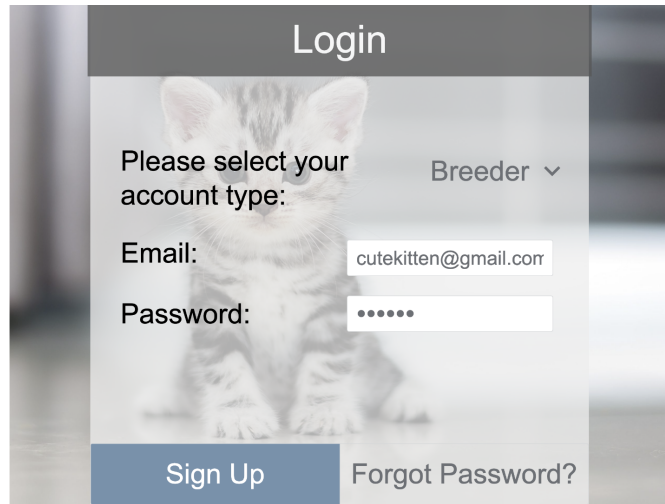
The method will close the questionnaire. All unsaved data will be deleted.

Appendix A: Database ER diagram design



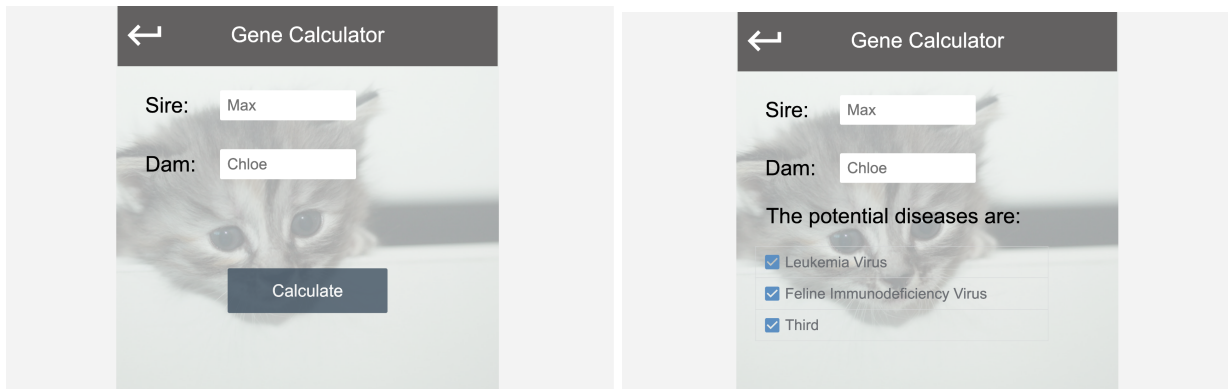
Appendix B: UI Design

B.1 Login Page:



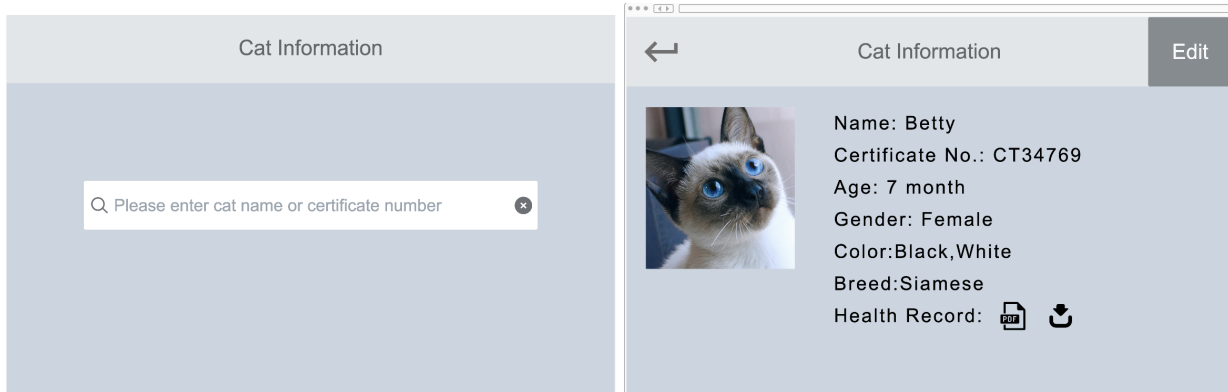
The login page features a dark header with the title "Login". Below the header, a light gray background contains a blurred image of a kitten. The text "Please select your account type:" is followed by a dropdown menu labeled "Breeder" with a downward arrow. Below this, there are two input fields: "Email:" with the value "cutekitten@gmail.com" and "Password:" with a masked input ".....". At the bottom, there are two buttons: "Sign Up" in a blue box and "Forgot Password?" in a gray box.

B.2 Gene Calculator:



The Gene Calculator page has a dark header with a back arrow and the title "Gene Calculator". Below the header, there are two input fields: "Sire:" with the value "Max" and "Dam:" with the value "Chloe". A "Calculate" button is positioned below these fields. The page then displays "The potential diseases are:" followed by a list of three items, each with a checked checkbox: "Leukemia Virus", "Feline Immunodeficiency Virus", and "Third".

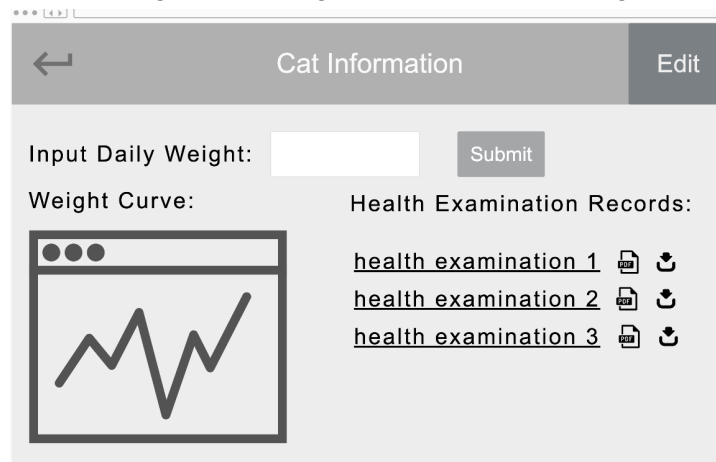
B.3 CatInfo:



The Cat Information page has a dark header with the title "Cat Information". Below the header, there is a search bar with the placeholder text "Q Please enter cat name or certificate number" and a magnifying glass icon. To the right of the search bar is a plus icon. The page then displays a cat's profile with a photo of a Siamese cat. The profile information includes: "Name: Betty", "Certificate No.: CT34769", "Age: 7 month", "Gender: Female", "Color:Black,White", "Breed:Siamese", and "Health Record:" followed by two icons representing a document and a download.


B.4 PregnantCatInfo:

Pregnant cats will have the same interface as other cats, except it contains a button to go to the pregnancy information. The pregnant cat page is shown as following:









Cat Information

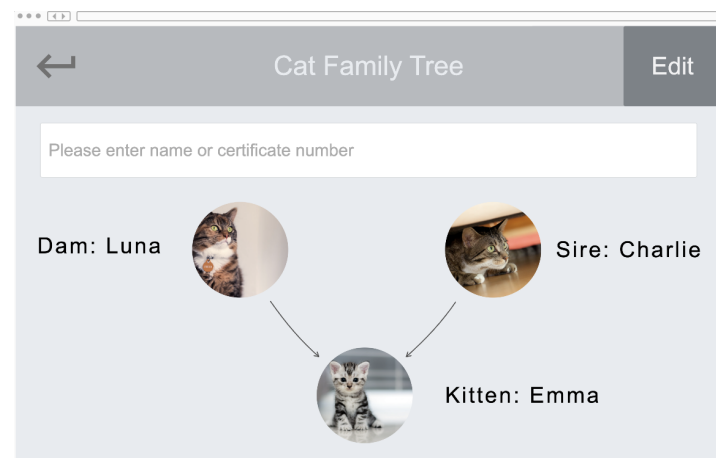
Input Daily Weight: Submit

Weight Curve: 

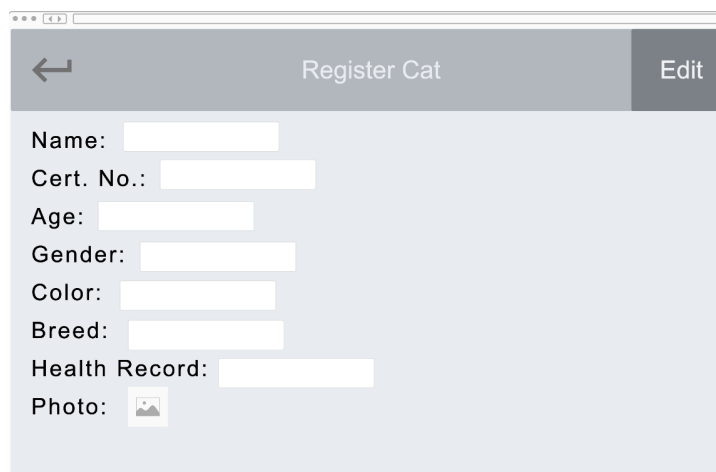
Health Examination Records:

- [health examination 1](#)  
- [health examination 2](#)  
- [health examination 3](#)  

B.5 Family Tree:



B.6 Register Cat:



Register Cat

Name:

Cert. No.:


Age:

Gender:

Color:

Breed:

Health Record:

Photo: 

Appendix C: References

[1] Software Requirements Specification for Web Cattery Management System (SRS)