Swipe *fur* Pets

University of Southern California
CSCI201 Summer 2020
Team Members: Mei Zhang, Hannah Oh, Jae Shim, Paing Kyaw, Xi Liu

TABLE OF CONTENTS

Introduction

1. Purpose

This document for “Swipe fur Pets” outlines the functional and nonfunctional requirements for the Web application built by the team.

2. Project Perspective

When you are looking for someone to have a romantic dinner within Santa Monica, you log onto Tinder or Bumble to choose a partner of your choice based on their age, sex, location, and looks. However, as many say, pets are better people. So why not have a Tinder-like Web application for adopting your perfect future pet? Swipe for Pets allows users to input their preferences, such as size, age, breed, sex, and others, to match them with adoption profiles. For example, when submitted that a user is looking for a Golden Retriever with weight ranging from 5 - 50 pounds and age between 1 - 3 year within the range of 30 miles of where they are located, the application would display adoption profile one by one, allowing the user to “swipe” to match or reject. Having an option to set up a specific range for weight or age allows the user to be matched with more specific profiles, rather than having to select among broad, subjective terms like “small, medium, or large.” The Web application aspect provides a more accessible, user-friendly environment, encouraging adoptions for many who were hesitant.

Software Requirements Specification

The program “Swipe *fur* Pets” will be a Web application that allows people who’re looking to adopt pets to swipe for pets. Users can sign up as adopters while shelters can sign up as adoptees. The specifications for different aspects of the application are listed below.

1. Authentication System Features

1.1 Login

a. ***Adopter-Specific Functional Requirements:***.An adopter can create an adopter-level account using two-factor authentication (email and phone number) in order to prevent spam account creation, after linking their email and password in a form

b. ***Adoptee-Specific Functional Requirements:*** An adoptee rescue organization can create an adoptee-level account with an organization identification number.

Several people can have access to this account, and every pet has their own profile

c. ***Technical Specifications:***

i. Implementation: 4 hours

ii. "User" and "Shelter" tables will be used in the database to store user information and rescue organization information for the purposes of storage and identification

1.2 Profile (for display)

a. ***Adopter-Specific Functional Requirements:*** A adopter can write a short description of themselves and their lifestyles along with up to 8 photos of their choice, such as their backyard or family.

b. ***Adoptee-Specific Functional Requirements:*** A adoptee rescue organization can post upto 8 photos of their pet along with a short description of their listing.

i. Examples: sleep habits, favorite food, favorite activities

c. Users can edit their profiles at a later time

d. ***Technical Specifications:***

i. Implementation: 2 hours

ii. HTML/CSS/Bootstrap to display all webpages. Information for user will be retrieved from respective "User" and "Shelter" table

iii. There will be a "Photos" table that stores all the photo images users upload and each photo will be associated with the user

1.3 Preferences (settings)

a. ***Adopter-Specific Functional Requirements:*** Adopter profiles are met with an empty form and able to input preferences for the type of pet they're looking for:

i. Animal specifications (species, breed, sex, age, size/weight, neutered, fur/hair, potty trained)

ii. Location radius

b. ***Adoptee-Specific Functional Requirements:*** Adoptee profiles see an empty input form and fill in preferences for the type of owners they desire for the pet

i. Child-free

ii. Living area details (house/apartment size, backyard size)

iii. Households with pre-existing pets

iv. Users can edit their preferences at a later time

c. ***Technical Specifications:***
    i. Implementation: 3 hours
    ii. HTML and Bootstrap will be used to display these forms and return this information to our Java servers
    iii. “Preferences” table will be used to store user preferences

1.4 Nonfunctional

Logging in should be automatic after the first time a user inputs it into their web and inputting preferences for each profile should take no more than 10 minutes.

2. Swiping, and Chatting

2.1 Swiping

a. ***Adopter-Specific Functional Requirements:*** An adopter swipes on pet listings that they are shown (based on their submitted preference) if they find the pet profile fitting to their needs/wishes.
b. ***Adoptee-Specific Functional Requirements:*** An adoptee selects on a specific listing they posted and swipes on adoptee profiles that wish to adopt
c. ***Keyboard Functionality:***
    i. **Right key:** Signals that the adopter/adoptee is interested in this pet/adopter.
    ii. **Left key:** Indicates that adopter/adoptee isn’t interested in this pet/adopter.
d. ***Technical Specifications***
    i. Implementation: 5 hours
    ii. "Swipe” animation happens when the left/right key is clicked
    iii. “User” table will be used to retrieve information and generate a list of recommendations based on user’ preferences.

2.2 Chatting

e. Once matched, potential adopters and adoptees are able to discuss further about the specific pet listing. The matching process denotes that the adopter and adoptee have matched, i.e., they both swiped right on each other.
f. ***Technical Specifications:***

i. Implementation: 5 hours
ii. HTML and Bootstrap will be used to display this UI for chatting interface; multithreading should also be implemented to allow the typing and sending message between both parties
iii. “Messages” table will be used to store messages and indicate User and Receiver

2.4 Nonfunctional

Displaying profiles should be displayed based on adopter’s distance preference (up to 60 miles) and should show how far they are from the adopter’s current location.

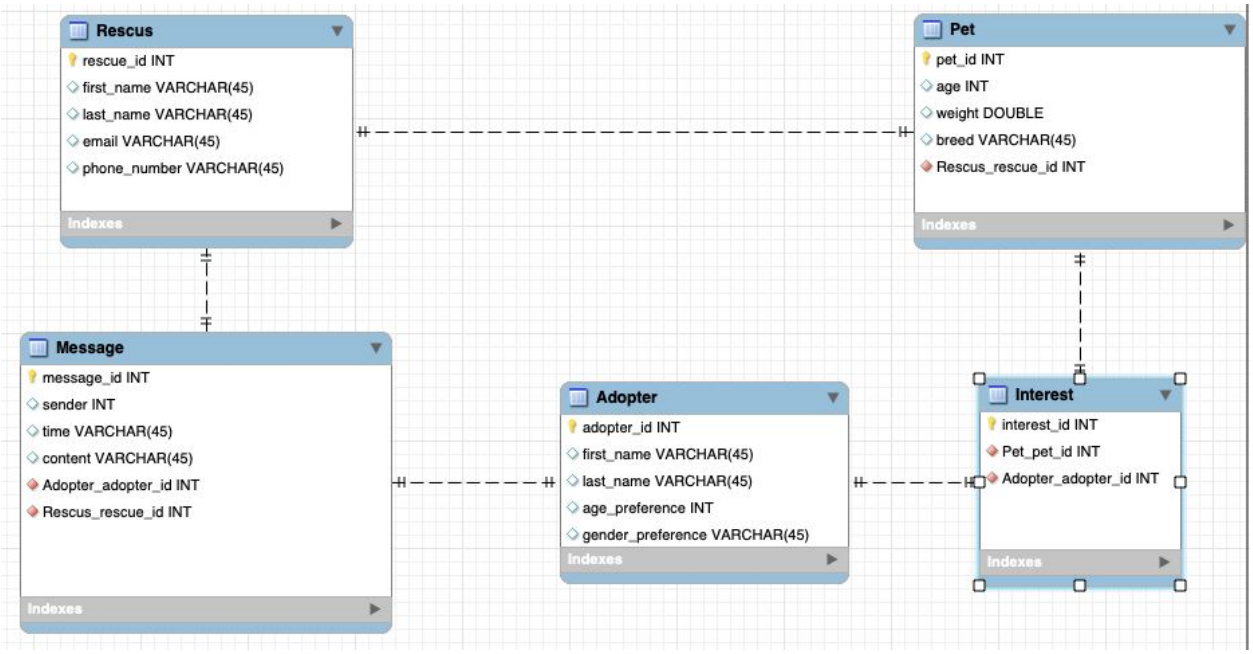
Constraints

Functional Constraints

- Functionality will be limited to the Web platform.
- An minimally viable product will be delivered for the first delivery. Expect unresolved issues and bugs.
- Ultimately, some functionalities planned for the final product may not be present in the deliverable for CSCI-201 due to time constraints.

Design Diagram

1. *Entity-Relationship Diagram*



Adopter:

- **UID (primary key)**
- first_name
- last_name
- email
- size_preference
- age_preference
- gender_preference

Pet:

- **PETID (primary key)**
- Rescue ID (foreign key)
- Age (int)
- Weight (double)
- Breed (String)
- Dog or cat (boolean)

Rescue:

- **ID (primary key)**
- first_name (String)
- last_name (String)
- email
- phone_number

Message:

- **ID (primary key)**
- Adopter (foreign key from Adopter)
- Rescue (foreign key from Rescue)
- Sender (boolean: 1 = adopter, 0 = Rescue sent this msg)
- TimeStamp
- Message Content

2. Class Diagram

a. Adopter Class

i. Data Members

1. fName: String
2. lName: String
3. email: String
4. password: String
5. phone : String
6. zip: Integer

ii. Preference Class

1. email: String
2. petType: String
3. size: String
4. age: String
5. children: Integer
6. dogs: Integer
7. cats: Integer

b. Rescue Class

i. Data Members

1. rescue_ID: String
2. organization_Name: String
3. email: String
4. phone: String
5. link: String
6. city: String
7. zipcode: Integer
8.

ii. Pet Class

1. pet_ID: Integer
2. pet_Name: String
3. petType: String
4. age: String
5. size: String
6. gender: String
7. rescue_ID: String
8. imgLink: String

c. Messages Class

i. Data Members
    1. message_ID: Integer
    2. sender: boolean
    3. pet_ID: Integer
    4. email: String
    5. time: Date
    6. message: String

Networking Functionality and Multithreading

- Multithreading functionality is included when a rescue indicates that they would like to set up a visit for the pet to be put up for adoption. This notification can appear at all times: whether the user is in the editing preferences screen or chatting with another pet, etc.
    - Multithreading functionality was not prepared to be present for our presentation due to challenges
- Petfinder API Support Documentation
    - Use Petfinder API in order to retrieve information regarding pets and rescues

Testing

For JUnit Testing:

Servlet: userLogin.java

Request: get

Parameters:

1. String requestType (registerUser or getUserProfile)
2. String userID

Expected Return:

If ***registerUser:***

String “Registered” or String “Not registered”

If getUserProfile:

ArrayList of String userID, firstName, fullName

Servlet: Adopter.java

Request: get

Parameters:

1. int firstName

Expected Return:

ArrayList of firstName, lastName, int agePref, String breedPref

Request: post

Parameters:

1. int petID

Expected Return:

None

Request: post

Parameters:

1. first name
2. last time
3. email
4. size_preference
5. age_preference

Expected Return:

None

Request: swipe

Parameters:

2. int petID

Expected Return:

Match ArrayList of adopter petID

Servlet: Rescue.java

Request: get

Parameters:

1. String rescueID

Expected Return:

ArrayList of String rescueID, String rescueName, String email, int phoneNum

Request: post

Parameters:

1. String rescueID

Expected Return:

None

Servlet: Pet.java

Request: get

Parameters:

1. int petID

Expected Return:

int age, double weight, String breed, int rescueID, Profile petProfile

Servlet: Message.java

Request: get

Parameters:

1. int message ID

Expected Return:

ArrayList of boolean sender, LocalDateTime time, String content, int adopterID, int petID

Servlet: Profile.html

Request: get

Parameters:

1. String rescue/petID

Test Cases

Test Case #1 (White Box): Register

- Test the login functionality by specifying a username or email that exists. The user should be taken back to the register page with a message "User already exists."

Test Case #2 (White Box): Login

- Test the login functionality by specifying a username that exists and a password that does not match. The user should be taken back to the login page with a message "Invalid login."

Test Case #3 (White Box): Swipe Right

- Test the swipe right functionality by swiping right on a profile. The user should have a visual cue to indicate the swipe has gone through indicated by a Heart <3

Test Case #5 (White Box): Swipe Left

- Test the swipe left functionality by swiping left on a profile. This means this profile was passed on and the next profile in the queue should be shown.

Test Case #6 (White Box): Create Profile

- Test the input fields to make sure users do not put in incorrect data types (for example, putting a number in first name or last name or leaving it empty)
- The screen should prompt the user to correct their mistakes

Test Case #7 (White Box): Fill in Preferences

- Test the input fields to make sure users do not put in incorrect data types
- The screen should prompt the user to correct their mistakes

Deployment

1. Ensure all java, js, and html/css files exist and are in the right folder.
2. Ensure gson library is imported/configured properly so the data can be parsed when using API calls from Pet Finder.
3. The Java version where the program is going to be running is in Java 8.
4. When running, make sure the port for the application is available (default port is 8080).

To run the front-end, type “cd directory/.../project_folder” in the terminal. Then, type “npm start” If encountering an error, type “unset HOST” to temporarily remove the local environment variable, then type “npm start” again. Now, the application should run on http://localhost:7000.