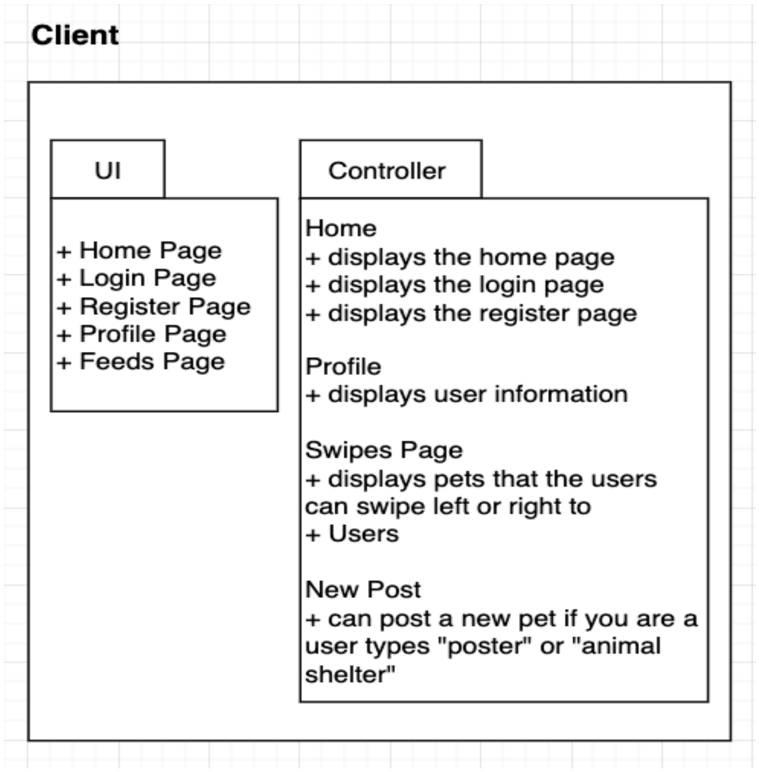
Furrble

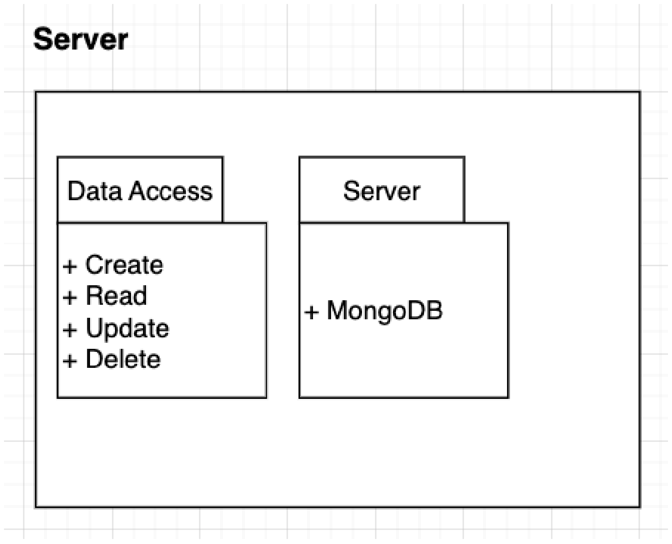
Daniel Floyd, Eikhi Ya Khaing, Logan Urch, Santiago Dubon, Kevin Avila

ITCS 4155
D2: Software Design Document

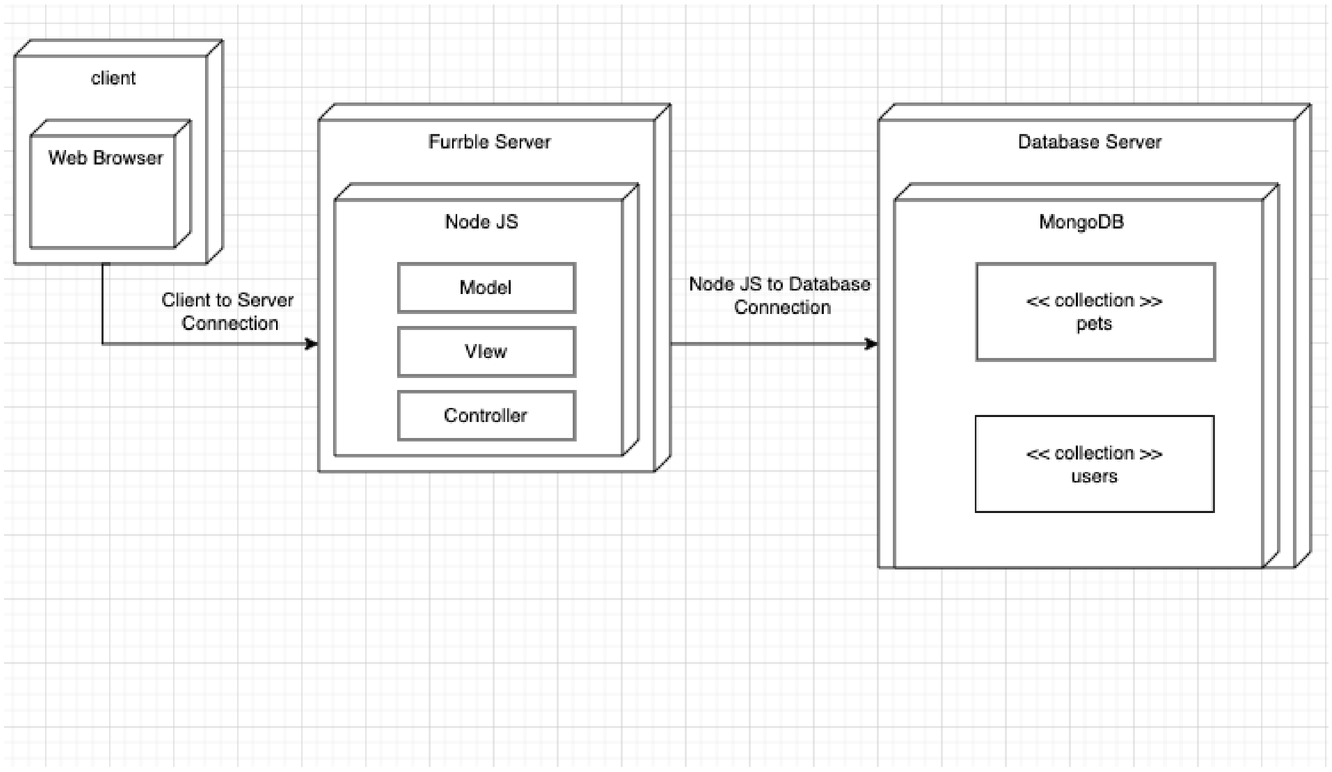
1. Project Overview
    a. This project aims to help solve the problem of overcrowded, unknown animals trapped in shelters around the globe. We decided a fun way to combat this is via a swiping system that many of today's apps use. Furrble is a project that we believe helps bridge the gap between potential adopters and pets looking for a forever home while being an enjoyable experience for the user. The primary stakeholders are potential adopters and the two parties listing animals: the individual posters and the shelters themselves. The stake that the posters have is helping their animals find a better home and the app allows them to very easily post them online for others to view. The stake that the swipers or default users have is that they are looking for a pet to complete their family and by using Furrble, they can easily find the perfect animal to do so.

2. Architectural Overview: As of right now, Furrble is designed to be used on a laptop or desktop computer. We are using a web based architecture to provide an easy to use platform for people looking to adopt pets. For our website, we used the node js MVC (model-view-controller) software architectural pattern. The model represents the data and business logic. For our website, we are using javascript to send and receive data from our database. The view represents the user interface and presentation layer. For our website, we are using embedded javascript to generate dynamic HTML pages by embedding javascript within HTML markup. The controller acts as an intermediary between the model and view to handle user requests and update the model as necessary. For our website, we are using javascript to process login information and authenticate users. The controller also handles displaying the profile page to the user.
    a. Subsystem Architecture
        i. Our application will be broken into two subsystems: the client layer and the server layer. The client layer handles all of the user-facing content, such as the account registration and the actual pet swiping. The server layer handles all of the data storage and access, such as account registrations, login requests, and pet database storage.


Client
UI
+ Home Page
+ Login Page
+ Register Page
+ Profile Page
+ Feeds Page
Controller
Home
+ displays the home page
+ displays the login page
+ displays the register page
Profile
+ displays user information
Swipes Page
+ displays pets that the users can swipe left or right to
+ Users
New Post
+ can post a new pet if you are a user types "poster" or "animal shelter"



b. Deployment Architecture
   i. The application is being deployed on the Heroku platform, while the database is being deployed on MongoDB Atlas. This architecture will allow for easy management and possible future scaling of the application and database separately.

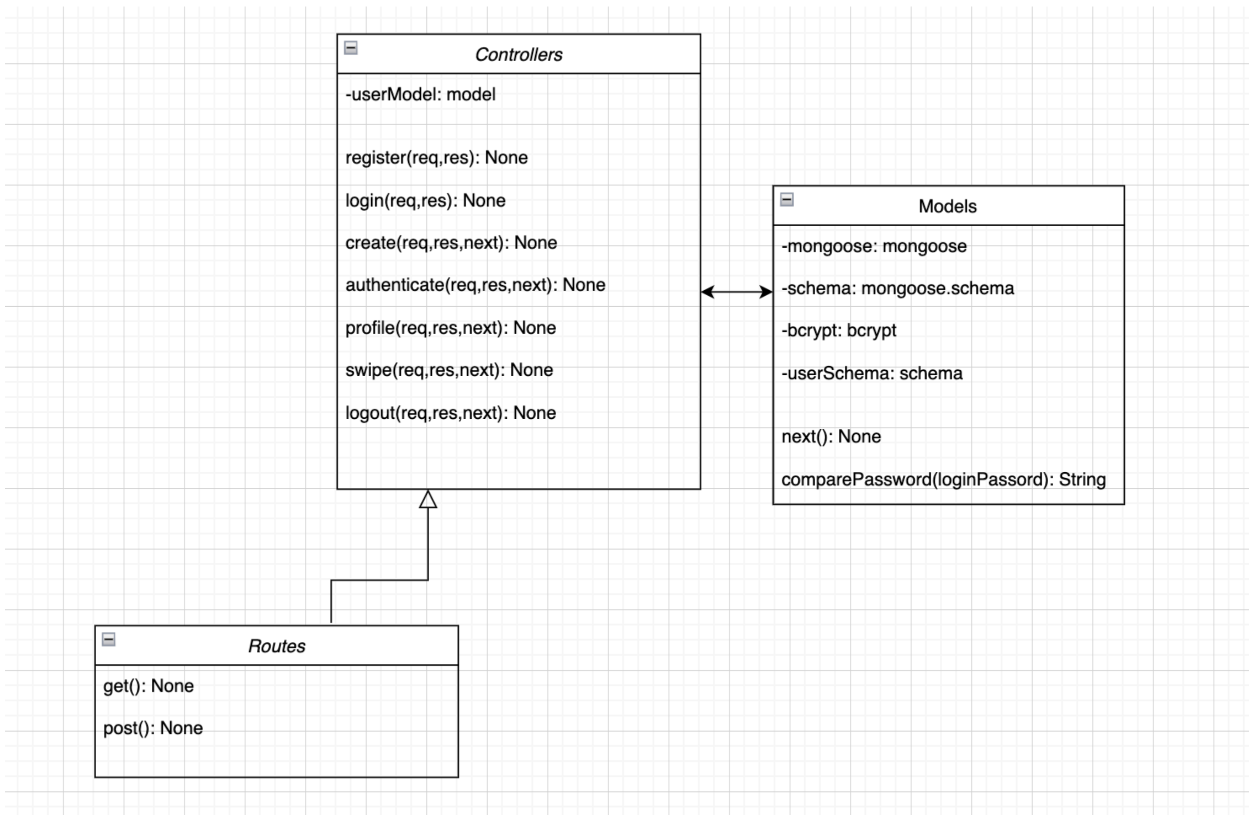


c. Persistent Data Storage

i. Our project will store data by using the MongoDB Atlas database. The collections that will be used are for user accounts and the pets themselves. The user accounts will use 6 columns with those being username, password, email, name, purpose for the use of the website, and liked pets. The username will have to be unique with no one else in the collection having that same username, and the same goes for the email. The password does not have to be unique since that would be a security issue for someone to know that a certain password is already being used. The name also does not have to be unique as there are multiple people in the world who have the same first and last name. In terms of purpose for the use of the website, this simply means if you are just any other individual or a shelter that's looking to post their pets on the site. The liked pets column will store all the pets using the pet ID from the pet collection that each specific user likes so that they never have to worry about hopping off the app and not finding them again. Users will also have the option to remove the pet from their liked pets and this will simply do exactly what it says which is removing the pet from the collection. The pet collection will have 4 columns with those being pet ID, species, breed, name. The first column will have to be unique as not more than one pet can or should have the same pet ID. With a pet ID they are easily identifiable, and you would not have to be worried about not being able to find them. This also makes it easier to add to the user accounts collection as having more than one of the same pet ID would be very confusing for both the user and the application. None of the last 3 columns will have to be unique within the collection as it is very possible to have more than one German Shepherd named Max.

    d. Global Control Flow
        i. Furrble is procedure driven. Every time a user opens the webpage, they are greeted with the homepage and a button that says “start swiping today” which takes them to the account creation/register form unless they were previously signed in. The actions that they can perform are based on what part of the application they are on. When on any of the forms (i.e. login, sign up) they can fill out those forms exactly as they are intended and are taken appropriately to the pages that follow each time. Once signed in, they are taken to the profile page where from there, they are greeted with their account information and can start swiping on animals or log back out. When on the swiping page, they always have the option to either swipe left or right on the pet by clicking and dragging either way. There is no time dependency or concurrency present in Furrble.

3. Detailed System Design
    a. Static View
        i. For our static view design we are going to use the MVC architecture and will have models of users (users will either be a poster/shelter or adopter) and pets for the MongoDB, additionally, there will be template EJS files to display the corresponding and correct information to the user. The EJS view directory will include many things such as homepage, about, or more user specific things such as login, signup, swiping, and profile pages.

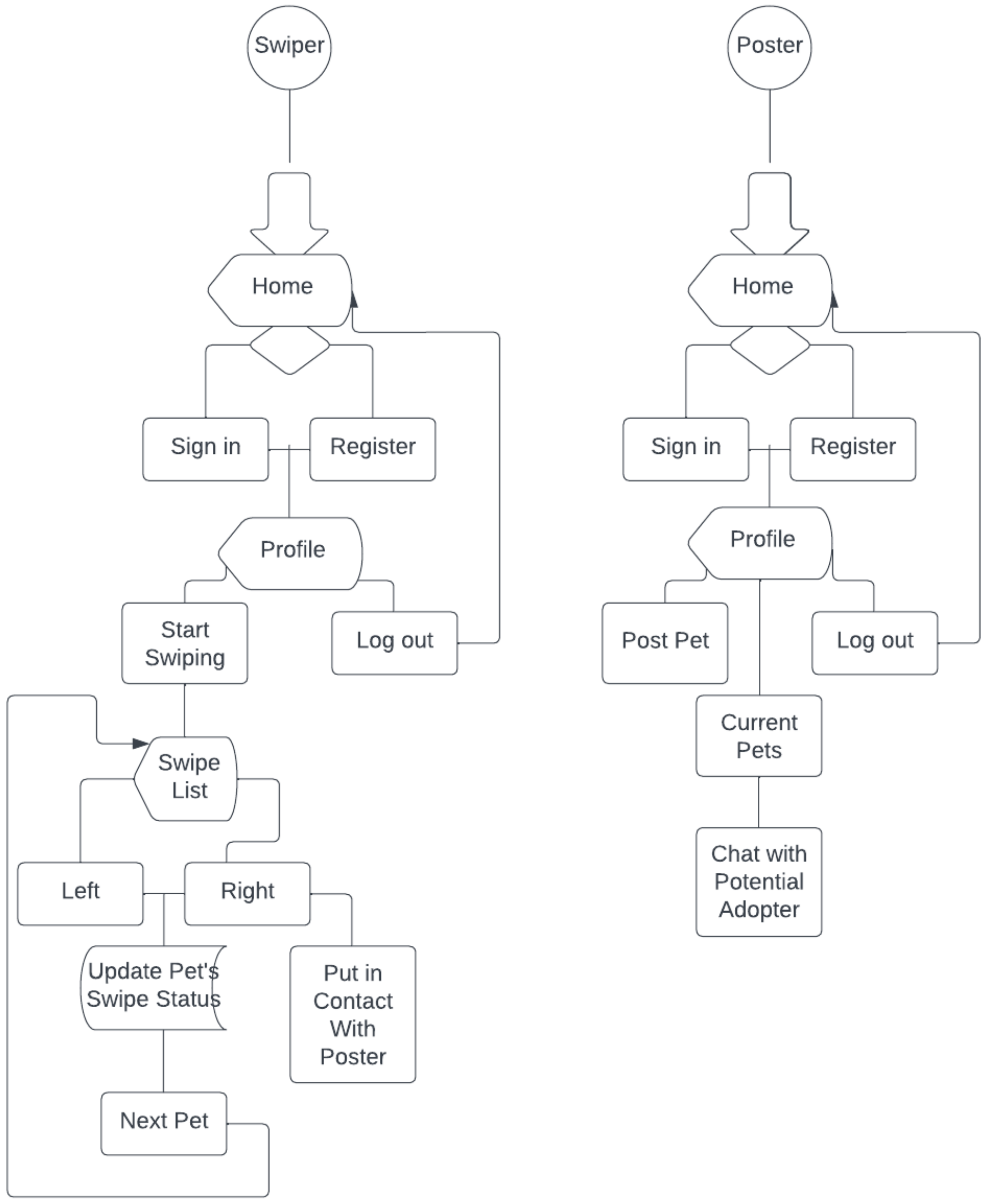


b.

c. Dynamic View

   i. For our dynamic view we are going to implement controllers, routes, and middleware functions which will help us with the CRUD functionality of the application as well as authorization to certain parts of the application. These items will also help us pull information from the MongoDB and help give the application the matching functionality as well. Finally, the controllers and routes will help us allow the user to access the website and enjoy the product.

   ii. Dynamic Diagram:


Swiper
Home
Sign in
Register
Profile
Start Swiping
Log out
Swipe List
Left
Right
Update Pet's Swipe Status
Put in Contact With Poster
Next Pet
Poster
Home
Sign in
Register
Profile
Post Pet
Log out
Current Pets
Chat with Potential Adopter