CSCC01

System Design Document

Rohan Dey, Ali Orozgani, Mohannad Moustafa Shehata, Vinesh Benny, Tarushi Thapliyal, Leila Cheraghi Seifabad

8th October, 2021

Table of Contents

CRC Cards

Backend:

Class Name: /api/User.js	
Parent/Subclass: None	
Responsibilities: • Hold methods for posting, querying, updating user data.	**Collaborators:** • /models/User.js

Class Name: /models/User.js	
Parent/Subclass: None	
Responsibilities: • Define the User model. In other words, how the User data is formatted to be sent to MongoDB.	**Collaborators:** • None

Class Name: /config/db.js	
Parent/Subclass: None	
Responsibilities: • Connect to the MongoDB database with MONGODB_URI in .env	**Collaborators:** • None

Class Name: Server.js	
Parent/Subclass: None	
Responsibilities: • Initialize and run the Express Node.js server.	**Collaborators:** • /config/db.js • /api/User.js

Frontend:

Class Name: App.js	
Parent/Subclass: None	
Responsibilities: ● The main function where the app runs from. Redirects responsibilities to RootStack.	**Collaborators:** ● /navigators/RootStack.js

Class Name: /components/KeyboardAvoidingWrapper.js	
Parent/Subclass: None	
Responsibilities: ● Prevents the keyboard from covering the text inputs as a User attempts to type on it.	**Collaborators:** ● /components/styles.js

Class Name: /components/styles.js	
Parent/Subclass: None	
Responsibilities: ● Holds all the created styles and colours. ● Makes it easier to call on predefined style templates to use in many different areas.	**Collaborators:** ● None

Class Name: /navigators/RootStack.js	
Parent/Subclass: None	
Responsibilities: ● Connect the screens together with React Navigation so one can navigate around the app.	**Collaborators:** ● /components/styles.js ● /screens/Login.js ● /screens/Signup.js ● /screens/Setting.js ● /screens/PetSitterMain.js ● /screens/AdminMain.js

	• /screens/PetOwnerMain.js

Class Name: /screens/Login.js	
Parent/Subclass: None	
Responsibilities: • Enter email address and password. Upon successful login, get redirected to the corresponding main page.	**Collaborators:** • /components/KeyboardAvoidingWrapper.js • /components/styles.js

Class Name: /screens/Signup.js	
Parent/Subclass: None	
Responsibilities: • Enter email address, full name, password, account type. Upon successful signup, get redirected to the corresponding main page.	**Collaborators:** • /components/KeyboardAvoidingWrapper.js • /components/styles.js

Class Name: /screens/Setting.js	
Parent/Subclass: None	
Responsibilities: • Update user and account information through the settings page.	**Collaborators:** • /components/KeyboardAvoidingWrapper.js • /components/styles.js

Class Name: /screens/PetSitterMain.js	
Parent/Subclass: None	
Responsibilities: • Act as a directory and have buttons that offer the Petsitter certain features such as “edit listing”.	**Collaborators:** • /components/styles.js

Class Name: /screens/AdminMain.js	
Parent/Subclass: None	
Responsibilities: ● Act as a directory and have buttons that offer the Admin certain features such as “manage users” and “manage listings”.	**Collaborators:** ● /components/styles.js

Class Name: /screens/PetOwnerMain.js	
Parent/Subclass: None	
Responsibilities: ● Act as a directory and have buttons that offer the Petsitter certain features such as “edit listing”.	**Collaborators:** ● /components/styles.js

Class Name: /screens/Services.js	
Parent/Subclass: None	
Responsibilities: ● Services page that the pet owner can access to view and scroll through listings.	**Collaborators:** ● /components/styles.js ● /components/Entry.js

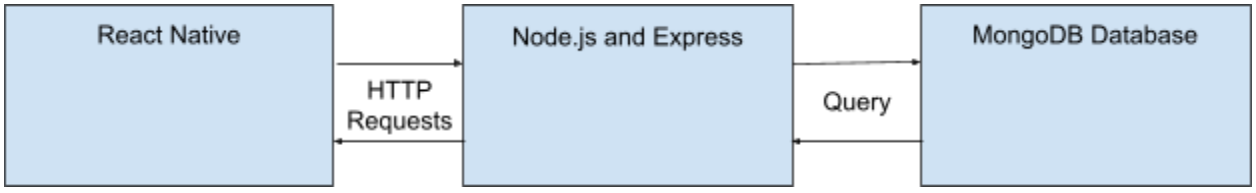
Class Name: /components/Entry.js	
Parent/Subclass: None	
Responsibilities: ● Represents the listing component boxes that contain each hosted service.	**Collaborators:** ● None

System Interaction Description

The Pawsup application assumes that the user has Node.js(^14.17.6) and npm(6.14.15) installed. The server is already being hosted on Heroku, but if one wishes to run it locally, they can cd into the backend-database folder, open their command line and type node server.js. The user should also be using either MacOS, Windows 7+ or Linux. As of now, the user is able to run the application by downloading all the required node modules provided in the package.json file by typing npm run install-all in the command line and then, typing expo start in the command line. You will require a web browser to run the application, but for the optimal experience, it is nice to have an AVD, iOS emulator or a physical device plugged in.

System Architecture

Our architecture resembles a typical mobile application that requires a three-tiered architecture. Our frontend uses React Native and connects to the backend with Axios HTTP requests. Our backend uses Node.js and Express and is hosted on a Heroku hosting server. It connects to the MongoDB database using JavaScript and one can perform queries to it. Below is an image showing how our system architecture looks like:



This in fact, follows the three-tiered architecture as there is the presentation tier with React Native, a logic tier with Node.js and Express and a Data tier where queries are sent to with MongoDB.

System Decomposition

Users will have access to the frontend React Native pages and they will have full functionality in terms of sending information to the backend and receiving information from it. An example of this is signing in and logging out of an account which will send the backend server an HTTP request. The server will then send a query to the MongoDB database and the database will send the data back to the server, which then sends it to the frontend, ultimately to the user.

Users receive error messages on the frontend when there is an issue and for the safety of other users, they do not have direct access to the database. This means that they cannot find out the password of another user by attempting to login incorrectly with their email.