CSCC01

# Documentation

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

8th October, 2021

# Table of Contents

# Backend

## backend-database/api/User.js:

backend-database/api/User.js is in essence the heart of the file in which it manages the server and does practically all of the backend functionality in terms of defining queries. It imports express and mainly utilizes the router method in it. The query methods are:

- /signup
    - This is used in pawsup-frontend/screens/Signup.js to create and save a new User to the database.
- /signin
    - This is used in pawsup-frontend/screens/Login.js to check if there exists a valid User in the database with email provided from Login, and if password matches what is stored. This essentially used to grant a user access to the rest of the app.
- /update
    - This is used in pawsup-frontend/screens/Settings.js to update a User's password and/or pettype.

The query methods are all defined below with more detail and example inputs and outputs:

### Create a user's profile:

#### Request

| | |
|---|---|
| Mimetype | application/json |
| Method | POST |
| URL | /api/user/signup |
| Description | It takes in a req and res parameter, where req holds the data of the user trying to sign up. Signup then ensures that the user does not already exist and all the parameters are of proper format. If the data is of the proper format defined and the user does not already exist, the data will then be sent to MongoDB. If that succeeds, res returns "SUCCESS" as its status, "Signup Successful" as its message and the User data as its data. Upon failure, res returns "FAILED" as its status, and the corresponding error as its message. |
| Body Example | {<br>email: "email",<br>password: "password",<br>fullname: "full name", |

| | |
|---|---|
| | dateofbirth: “YYYY/MM/DD”,<br>phonenumber: “xxxxxxxxxx”,<br>accounttype: “accounttype”,<br>pettype: “pettype”<br>} |

### Response

| | |
|---|---|
| Mimetype | application/json |
| Body Example: Individual | {<br>"status": "SUCCESS",<br>"message": "Signup Successful",<br>"data": {<br>"email": "email@t.com",<br>"password": "password",<br>"fullname": "full name",<br>"dateofbirth": "2001/10/20",<br>"phonenumber": "1010101010",<br>"accounttype": "Petsitter",<br>"pettype": "Dog",<br>"_id": "615fad6612ed1eaae5f942d3",<br>"__v": 0<br>}<br>} |

## Sign in using a user’s credentials:

### Request

| | |
|---|---|
| Mimetype | application/json |
| Method | POST |
| URL | /api/user/signin |
| Description | It takes in a req and res parameter, where req holds the email and password of the user trying to sign in. Signin then ensures that the user truly exists and whether all the parameters are nonempty. If the data is of the proper format defined and the user exists, the data will then be sent to MongoDB. If that succeeds, res returns “SUCCESS” as its status, “Signin Successful” as its message and the User data that is retrieved is sent as its data. Upon failure, res returns “FAILED” as its status, and the |

| | |
|---|---|
| | corresponding error as its message. |
| Body Example | {<br>email: “email”,<br>password: “password”,<br>} |

### Response

| | |
|---|---|
| Mimetype | application/json |
| Body Example: Individual | {<br>"status": "SUCCESS",<br>"message": "Signin Successful",<br>"data": {<br>"email": "email@t.com",<br>"password": "password",<br>"fullname": "full name",<br>"dateofbirth": "2001/10/20",<br>"phonenumber": "1010101010",<br>"accounttype": "Petsitter",<br>"pettype": "Dog",<br>"_id": "615fad6612ed1eaae5f942d3",<br>"__v": 0<br>}<br>} |

## Update a user’s password and/or pettype:

### Request

| | |
|---|---|
| Mimetype | application/json |
| Method | PUT |
| URL | /api/user/update |
| Description | It takes in a req and res parameter, where req holds the email, password and pettype of the user that is supposed to be updated. Update then ensures that the user has changed parameters and checks whether all the parameters are nonempty. If the data is of the proper format defined and the user wants to change some sort of data, the new data will then be sent to MongoDB. If that succeeds and the user is updated, res returns “SUCCESS” as its status, “Update Successful” as its message |

| | |
|---|---|
| | and the User data that is retrieved is sent as its data. Upon failure, res returns “FAILED” as its status, and the corresponding error as its message. |
| Body Example | {<br>email: “email”,<br>password: “password”,<br>pettype: “pettype”<br>} |

### Response

| | |
|---|---|
| Mimetype | application/json |
| Body Example: Individual | {<br>"status": "SUCCESS",<br>"message": "Update Successful",<br>"data": {<br>"email": "email@t.com",<br>"password": "password",<br>"fullname": "full name",<br>"dateofbirth": "2001/10/20",<br>"phonenumber": "1010101010",<br>"accounttype": "Petsitter",<br>"pettype": "Dog",<br>"_id": "615fad6612ed1eaae5f942d3",<br>"__v": 0<br>}<br>} |

## backend-database/config/db.js:

backend-database/db.js connects the server to the MongoDB database with the help from Mongoose and the MongoDB URL. The MongoDB URL is located in a dotenv file so that it remains hidden from the general public. Upon success, it will return “Database Connected”, but upon failure, it will catch and throw an error that tells the server that there was an issue.

## backend-database/models/User.js:

backend-database/models/User.js defines the User model with Mongoose Schema to present to the server and MongoDB the format of the User. It imports Mongoose and each User model takes in an email, password, full name, date of birth, phone number, account type and pet type that are all of type String. An example of a User model follows along with how it would be called and utilized:

```
const newUser = new User({
        email,
        password,
        fullname,
        dateofbirth,
        phonenumber,
        accounttype,
        pettype
});
```

## backend-database/Procfile:

backend-database/Procfile is a file that tells the Heroku Cloud Server how to use the created Node.js and Express server. In essence, it simply tells Heroku to run the command node server.js which would typically be run on a local server through the Command Line.

## backend-database/server.js:

backend-database/server.js defines the dependencies and calls the functions defined in backend-database/User.js when doing server requests. It sets which port for the server to be run on, and calls backend-database/config/db.js which connects to the database. It must be run firstly using node server.js in the backend-database folder before any database requests can be made.

# Frontend

## pawsup-frontend/screens/Setting.js:

pawsup-frontend/screens/Setting.js is the frontend screen that provides a given user the ability to change personal and account information. This screen imports icons from @expo/vector-icons and ultimately returns a display of the settings page. This page takes in 3 text inputs (new phone number, new password and new pet) and a user can submit this data with the button that says “Change Information”. Errors are sent to the user interface from the backend. Methods in this file include: handleSignup, handleMessage and MyTextInput. The functionality of each method is as follows:

- handleSignup
  - handleSignup handles the new data to be updated. It will send the data to the update query method defined in the backend.
- handleMessage
  - Upon failure while sending the update, handleMessage will output this message and tell the user something went wrong.
- MyTextInput
  - MyTextInput defines a style for the text input fields in which it has a left icon along with placeholder text. It then replaces the placeholder text with the inputted text upon received input. It also offers customization to show and hide the password.

## pawsup-frontend/screens/Signup.js:

pawsup-frontend/screens/Signup.js is the frontend screen that provides a potential given user the ability to become a user of the Pawsup app. This screen imports icons from @expo/vector-icons and ultimately returns a display of the signup page. This page takes in 7 text inputs (email address, password, full name, date of birth, phone number, account type and pet type) and a user can submit this data with the button that says “Signup”. Errors are sent to the user interface from the backend. Some example errors include the password being too short or the email missing an @ symbol. Methods in this file include: onChange, showDatePicker, handleSignup, handleMessage and MyTextInput. The functionality of each method is as follows:

- onChange
  - Holds the date selected on the calendar. Upon selection, it also closes DatePicker.
- showDatePicker
  - When the date field is clicked on the UI, this method runs to display the date picker. It allows the user to select the date.
- handleSignup
  - handleSignup handles the new data to be entered into the database. It will send the data to the signup query method defined in the backend.
- handleMessage
  - Upon failure while sending the signup request, handleMessage will output this message and tell the user something went wrong.

- MyTextInput
    - MyTextInput defines a style for the text input fields in which it has a left icon along with placeholder text. It then replaces the placeholder text with the inputted text upon received input. It also offers customization to show and hide the password.

## pawsup-frontend/screens/Login.js:

pawsup-frontend/screens/Login.js is the frontend screen that provides a given user the ability to enter the Pawsup app. This screen imports icons from @expo/vector-icons and ultimately returns a display of the login page. This page takes in 2 text inputs (email address and password) and a user can submit this data with the button that says “Login”. Errors are sent to the user interface from the backend. Some example errors include the password being incorrect or the email missing an @ symbol. Methods in this file include: handleLogin, handleMessage and MyTextInput. The functionality of each method is as follows:

- handleLogin
    - handleLogin handles the entered data. It sends the data to the login query method defined in the backend and we retrieve the data upon success.
- handleMessage
    - Upon failure while sending the login request, handleMessage will output this message and tell the user something went wrong.
- MyTextInput
    - MyTextInput defines a style for the text input fields in which it has a left icon along with placeholder text. It then replaces the placeholder text with the inputted text upon received input. It also offers customization to show and hide the password.

## pawsup-frontend/screens/PetSitterMain.js:

pawsup-frontend/screens/PetSitterMain.js is the frontend screen that provides a given logged in/signed up pet sitter the ability to enter the Pawsup app. This screen ultimately returns a display of the main directory for the pet sitter. This page has 4 buttons along with a settings icon. The settings icon redirects the user to the account settings page, and the 4 buttons are “create new listing”, “edit your listing”, “store”, and “your orders”. The purpose of each button is as follows:

- Create New Listing
    - Navigates the pet sitter to a page where they can create a listing with required data.
- Edit Your Listing
    - Navigates the pet sitter to a page where they can modify their listings.
- Store
    - Navigates the pet sitter to the store page where they can purchase items that they want.
- Your Orders

    - Navigates the pet sitter to their orders page where they can see which of their listings have been booked and what purchases they have made.

## pawsup-frontend/screens/PetOwnerMain.js:

pawsup-frontend/screens/PetOwnerMain.js is the frontend screen that provides a given logged in/signed up petowner the ability to enter the Pawsup app. This screen ultimately returns a display of the main directory for the petowner. This page has 3 buttons along with a settings icon. The settings icon redirects the user to the account settings page, and the 3 buttons are “services”, “store”, and “your orders”. The purpose of each button is as follows:

- Services
    - Navigates the petowner to the services page where they can scroll and view listings created by pet sitters.
- Store
    - Navigates the petowner to the store page where they can purchase items that they want.
- Your Orders
    - Navigates the petowner to their orders page where they can see what purchases they have made, whether it be a listing or from the store.

## pawsup-frontend/screens/AdminMain.js:

pawsup-frontend/screens/AdminMain.js is the frontend screen that provides a given admin the ability to enter the Pawsup app. This screen ultimately returns a display of the main directory for the admin. This page has 3 buttons along with a settings icon. The settings icon redirects the user to the account settings page, and the 3 buttons are “manage users”, “manage listings”, and “manage store products”. The purpose of each button is as follows:

- Manage Users
    - Redirects the admin to a page where they can moderate non-admin users. An example of its use is that it can be used to ban an irresponsible pet sitter from posting listings.
- Manage Listings
    - Navigates the admin to a page where they can administer different listings.
- Manage Store Products
    - Navigates the admin to a page where they can add, delete, or modify shop items.

## pawsup-frontend/components/styles.js:

pawsup-frontend/components/styles.js is the component page that used to style the Pawsup app. It has a variety of defined style methods. The purpose of each method is as follows:

- Colours
  - Stores the HTML colour codes that are needed to add colour to the UI.
- StyledContainer, InnerContainer
  - Custom containers with minor differences for formatting the UI of pages.
- PageTitle, SubTitle, StyledTextInput, StyledInputLabel, ButtonText, MsgBox, ExtraText, TextLinkContent
  - Custom images with different sizes for different pages.
- LeftIcon, RightIcon
  - Custom icon formatting for the text input boxes.
- RightIcon
  - Navigates the admin to a page where they can add, delete, or modify shop items.
- StyledButton
  - The format for the buttons on the pages.
- Line
  - Adds a horizontal line across the page UI.

## pawsup-frontend/components/KeyboardAvoidingWrapper.js:

pawsup-frontend/components/KeyboardAvoidingWrapper.js is the component page that implements an epic for the Pawsup app UI. This wrapper ensures that the keyboard never covers the text input field a user is trying to fill out so that the user can see what they are typing into the text input field. It implements the primary colour from Colours in pawsup-frontend/components/styles.js.

## pawsup-frontend/navigators/RootStack.js:

pawsup-frontend/navigators/RootStack.js is the navigator page that Pawsup app UI. This handles all the navigation definitions in the application. It imports every screen in the app and connects it to a constant and then adds it to a Stack Navigator so each screen can access each other fairly easily. It also implements darkLight, brand, primary, tertiary, secondary colours from Colours in pawsup-frontend/components/styles.js.