

CSCC01

Documentation

Rohan Dey, Ali Orozgani, Mohannad Moustafa Shehata, Vinesh
Benny, Tarushi Thapliyal, Leila Cheraghi Seifabad
21st October, 2021

Table of Contents

Table of Contents	1
Backend	3
backend-database/api/Router.js:	3
backend-database/config/db.js:	17
backend-database/models/User.js:	17
backend-database/models/Listing.js:	17
backend-database/Procfile:	18
backend-database/server.js:	18
Frontend	18
pawsup-frontend/screens/Setting.js:	18
pawsup-frontend/screens/Signup.js:	19
pawsup-frontend/screens/Login.js:	19
pawsup-frontend/screens/PetSitterMain.js:	20
pawsup-frontend/screens/PetOwnerMain.js:	20
pawsup-frontend/screens/AdminMain.js:	21
pawsup-frontend/screens/Services.js:	21
pawsup-frontend/screens/PetSitterModifyListing.js:	21
pawsup-frontend/screens/UpcomingAppointment.js:	22
pawsup-frontend/screens/DetailedListing.js:	22
pawsup-frontend/components/styles.js:	22
pawsup-frontend/components/KeyboardAvoidingWrapper.js:	23
pawsup-frontend/components/Entry.js:	23
pawsup-frontend/navigators/RootStack.js:	23

Backend

backend-database/api/Router.js:

backend-database/api/Router.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.
- `/getUser`
 - This is used in various frontend screens to retrieve the User data. It requires an email of a user and if the query goes through, it will provide the application with the corresponding User data.
- `/update`
 - This is used in pawsup-frontend/screens/Settings.js to update a User's password and/or pettype.
- `/createListing`
 - This is used in pawsup-frontend/screens/Signup.js to create a listing when told to create a petsitter account.
- `/getListing`
 - This is used in various frontend screens to get a petsitter's listing by their email.
- `/modifyListing`
 - This is used in pawsup-frontend/screens/PetSitterModifyListing.js to update the listing of the petsitter.
- `/makeBooking`
 - This is used in various frontend screens to either create blocked dates by the Petsitter or create booked dates by the petowner.
- `/cancelBooking`
 - This is used in various frontend screens to cancel an appointment either by a petowner, petsitter or admin.
- `/getPetownerBookings`
 - This is used in a frontend screen to get the bookings from the petowners so that they can see their upcoming bookings.
- `/filterPriceListing`
 - This is used in pawsup-frontend/screens/Services.js to filter listings by its price.

- [/filterAvailabilityListing](#)
 - This is used in pawsup-frontend/screens/Services.js to filter listings by its availability on a set of dates.

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/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	<pre>{ email: "email", password: "password", fullname: "full name", dateofbirth: "YYYY/MM/DD", location: "Street # Street Name, City, Province, Postal Code", phonenumber: "xxxxxxxxxx", accounttype: "accounttype", pettype: "pettype" }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Signup Successful",</pre>

	<pre> "data": { "email": "email@t.com", "password": "password", "fullname": "full name", "dateofbirth": "2001/10/20", "location": "123 Test Ave, Toronto, ON, M2C4P7", "phonenummer": "1010101010", "accounttype": "Petsitter", "pettype": "Dog", "_id": "615fad6612ed1eaae5f942d3", "__v": 0 } </pre>

Sign in using a user's credentials:

Request

Mimetype	application/json
Method	POST
URL	/api/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	<pre> { email: "email", password: "password", } </pre>

Response

Mimetype	application/json
----------	------------------

Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Signin Successful", "data": { "email": "email@t.com", "password": "password", "fullname": "full name", "dateofbirth": "2001/10/20", "location": "123 Test Ave, Toronto, ON, M2C4P7", "phonenumber": "1010101010", "accounttype": "Petsitter", "pettype": "Dog", "__id": "615fad6612ed1eaae5f942d3", "__v": 0 } }</pre>
-----------------------------	--

Get user's information:

Request

Mimetype	application/json
Method	GET
URL	/api/getUser
Description	It takes in a req and res parameter, where req holds the email of the user whose data is attempted to be retrieved. getUser 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, "User Found Successfully" 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	<pre>{ email: "email" }</pre>

Response

Mimetype	application/json
----------	------------------

Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "User Found Successfully", "data": { "email": "email@t.com", "password": "password", "fullname": "full name", "dateofbirth": "2001/10/20", "location": "123 Test Ave, Toronto, ON, M2C4P7", "phonenumber": "1010101010", "accounttype": "Petsitter", "pettype": "Dog", "_id": "615fad6612ed1eaae5f942d3", "__v": 0 } }</pre>
-----------------------------	---

Update a user's password and/or pettype:

Request

Mimetype	application/json
Method	PUT
URL	/api/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	<pre>{ email: "email", password: "password", pettype: "pettype" }</pre>

	}
--	---

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Update Successful", "data": { "email": "email@t.com", "password": "password", "fullname": "full name", "dateofbirth": "2001/10/20", "location": "123 Test Ave, Toronto, ON, M2C4P7", "phonenummer": "1010101010", "accounttype": "Petsitter", "pettype": "Dog", "_id": "615fad6612ed1eaae5f942d3", "__v": 0 } }</pre>

Create a listing for a Petsitter:

Request

Mimetype	application/json
Method	POST
URL	/api/createListing
Description	<p>It takes in a req and res parameter, where req holds the email of the pet sitter. createListing then checks whether the parameter is nonempty. If the data is of the proper format defined and no listing is associated with the provided email, a new empty listing will be made in MongoDB. If that succeeds and listing is created, res returns "SUCCESS" as its status, "Listing Creation Successful" as its message and the Listing data that is stored in MongoDB is sent as its data. Upon failure, res returns "FAILED" as its status, and the corresponding error as its message.</p>
Body Example	{

	<pre> listingowner: "email@t.com" } </pre>
--	--

Response

Mimetype	application/json
Body Example: Individual	<pre> { "status": "SUCCESS", "message": "Listing Creation Successful", "data": { "listingowner": "email@t.com", "title": "Not filled out yet", "location": "Not filled out yet", "features": "Not filled out yet", "price": -1, "bookings": [], "_id": "615fad6612ed1eaae5f942d3", "__v": 0 } } </pre>

Get a listing for a Petsitter:

Request

Mimetype	application/json
Method	GET
URL	/api/getListing
Description	<p>It takes in a req and res parameter, where req holds the email of the pet sitter. getListing then checks whether the parameter is nonempty. If the data is of the proper format defined and a listing is associated with the provided email, the listing is fetched from MongoDB. If that succeeds, res returns "SUCCESS" as its status, "Listing Found Successfully" as its message and the Listing data that is retrieved is sent as its data. Upon failure, res returns "FAILED" as its status, and the corresponding error as its message.</p>
Body Example	<pre> { listingowner: "email@t.com" } </pre>

	}
--	---

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Listing Found Successfully", "data": { "listingowner": "email@t.com", "title": "Not filled out yet", "location": "Not filled out yet", "features": "Not filled out yet", "price": -1, "bookings": [], "_id": "615fad6612ed1eaae5f942d3", "__v": 0 } }</pre>

Modify a listing for a Petsitter:

Request

Mimetype	application/json
Method	PUT
URL	/api/modifyListing
Description	<p>It takes in a req and res parameter, where req holds the email of the pet sitter, title, description, location, features and price of the new listing. modifyListing then checks whether the parameters are nonempty. If the data is of the proper format defined and a listing is associated with the provided email, the listing will be updated in MongoDB with the new info. If that succeeds and listing is created, res returns "SUCCESS" as its status, "Listing Modification Successful" as its message and the Listing data that is stored in MongoDB is sent as its data. Upon failure, res returns "FAILED" as its status, and the corresponding error as its message.</p>
Body Example	{

	<pre> listingowner: "email@t.com", title: "title", description: "description", location: "location", features: "features", price: "10" } </pre>
--	---

Response

Mimetype	application/json
Body Example: Individual	<pre> { "status": "SUCCESS", "message": "Listing Modification Successful", "data": { "listingowner": "email@t.com", "title": "title", "location": "location", "features": "features", "price": 10, "bookings": [], "_id": "615fad6612ed1eaae5f942d3", "__v": 0 } } </pre>

Make a booking for a Petsitter:

Request

Mimetype	application/json
Method	PUT
URL	/api/makeBooking
Description	<p>It takes in a req and res parameter, where req holds the email of the pet sitter, reason, cost, start date and end date of the booking. Reason can either be "BLOCKED", meaning it has been blocked by the pet sitter; or it can be the email of a pet owner, meaning that owner has made the booking. modifyListing then checks whether the parameters are nonempty. If the data is of the proper format defined and the provided</p>

	start and end date do not overlap with any bookings already made, the listing gets updated with the added booking in MongoDB. If that succeeds and booking is created, res returns "SUCCESS" as its status, "Listing Booking Added Successfully" as its message and the Listing data that is stored in MongoDB is sent as its data. Upon failure, res returns "FAILED" as its status, and the corresponding error as its message.
Body Example	<pre>{ listingowner: "email@t.com", reason: "BLOCKED", cost: "10", startdate: "2021/10/23", enddate: "2021/10/27" }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Listing Booking Added Successfully", "data": { "listingowner": "email@t.com", "title": "title", "location": "location", "features": "features", "price": 10, "bookings": [{ "reason": "BLOCKED", "cost": "10", "startdate": "2021/10/23", "enddate": "2021/10/27", "_id": "6170e8759ad0878171e61756" }], "_id": "615fad6612ed1eaae5f942d3", "__v": 0 } }</pre>

Cancel a booking for a Petsitter:

Request

Mimetype	application/json
Method	PUT
URL	/api/cancelBooking
Description	It takes in a req and res parameter, where req holds the email of the pet sitter, start date and end date of the booking. cancelListing then checks whether the parameters are nonempty. If the data is of the proper format defined, cancelBooking looks through bookings of the associated listing of email and checks if there are any bookings that match the provided start date and end date. If that succeeds, the booking is deleted, res returns "SUCCESS" as its status, "Listing Booking Removed Successfully" as its message and the Listing data that is stored in MongoDB is sent as its data. Upon failure, res returns "FAILED" as its status, and the corresponding error as its message.
Body Example	<pre>{ listingowner: "email@t.com", startdate: "2021/10/23", enddate: "2021/10/27" }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Listing Booking Removed Successfully", "data": { "listingowner": "email@t.com", "title": "title", "location": "location", "features": "features", "price": 10, "bookings": [], "_id": "615fad6612ed1eaae5f942d3", "__v": 0 } }</pre>

Get Petowner's Bookings:

Request

Mimetype	application/json
Method	PUT
URL	/api/getPetownerBookings
Description	It takes in a req and res parameter, where req holds the email of the petowner. getPetownerBookings then checks whether the parameters are nonempty. If the data is of the proper format defined, getPetownerBookings looks through bookings of all the listings and checks if there are any bookings with the petowner email. If that succeeds, a list with all bookings is created, res returns "SUCCESS" as its status, "Bookings Found Successfully" as its message and the Listing data that is stored in MongoDB is sent as its data. Upon failure, res returns "FAILED" as its status, and the corresponding error as its message.
Body Example	<pre>{ petowner: "testuser3@gmail.com" }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Bookings Found Successfully", "data": [] }</pre>

Filter listing by price:

Request

Mimetype	application/json
Method	GET

URL	/api/filterPriceListings
Description	It takes in a req and res parameter, where req holds the minimum and the maximum price that the endpoint should filter listings by. filterPriceListings then checks whether the parameters are greater than or equal to 0. If the data is of the proper format defined, filterPriceListings looks through all the listings and their prices. Anything that falls in between the minimum and maximum prices is added to a list. If the query succeeds, res returns "SUCCESS" as its status, "Listing Owners With Suitable Price Found Successfully" as its message and a list of listing owner emails that is stored in MongoDB is sent as its data. Upon failure, res returns "FAILED" as its status, and the corresponding error as its message.
Body Example	<pre>{ minprice: 12, maxprice: 17 }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Listing Owners With Suitable Price Found Successfully", "data": ["test@gmail.com"] }</pre>

Filter listing by availability:

Request

Mimetype	application/json
Method	GET
URL	/api/filterAvailabilityListings
Description	It takes in a req and res parameter, where req holds the start date and the end date that the endpoint should filter listings by. filterAvailabilityListings then reformats the dates and checks if the end

	date is earlier than the start date. If the data is of the proper format defined, filterAvailabilityListings looks through all the listings and their booking dates. Anything that does not have the dates in between start date and end date inclusive booked is then added to the list of emails. If the query succeeds, res returns "SUCCESS" as its status, "Listing Owners With Suitable Availability Found Successfully" as its message and a list of listing owner emails that is stored in MongoDB is sent as its data. Upon failure, res returns "FAILED" as its status, and the corresponding error as its message.
Body Example	<pre>{ startdate: 2021/10/19, enddate: 2021/10/21 }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Listing Owners With Suitable Availability Found Successfully", "data": ["test@gmail.com"] }</pre>

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, location, 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,
  location,
  phonenumber,
  accounttype,
  pettype
});
```

backend-database/models/Listing.js:

backend-database/models/Listing.js defines the Listing model with Mongoose Schema to present to the server and MongoDB the format of the Listing. It imports Mongoose and each Listing model takes in a listing owner's email, title, description, location, features, prices and bookings where price is a Number and bookings is a BookedSchema list with the fields, reason, cost, startdate and enddate of types String, Number, String and String respectively. The rest of the fields in the Listing schema are also all Strings. An example of a Listing model follows along with how it would be called and utilized:

```
const newListing = new Listing({
  listingowner,
  title,
  description,
  location,
  features,
  price,
  bookings
});
```

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/Router.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/screens/Services.js:](#)

pawsup-frontend/screens/Services.js is the frontend screen that provides the petowner with the ability to sort and filter through listings to find the ideal listing to book. This handles the visualization of all the listings on the database. It also implements various required colours from Colours in pawsup-frontend/components/styles.js along with other styles. Furthermore, it incorporates the Entry.js component file which assists with the UI in terms of columns and formatting.

[pawsup-frontend/screens/PetSitterModifyListing.js:](#)

pawsup-frontend/screens/PetSitterModifyListing.js is the frontend screen that provides the petsitter with the ability to create and edit their listing. This allows the petsitter to choose a title, description, location, features and price. It also allows them to block off dates on which they may be busy. This is accessed through PetSitterMain.js and it also implements darkLight, brand, primary, tertiary, secondary colours from Colours in pawsup-frontend/components/styles.js.

[pawsup-frontend/screens/UpcomingAppointment.js:](#)

pawsup-frontend/screens/UpcomingAppointments.js is the frontend screen that allows the user to see their upcoming appointments. The view differs depending on whether the user is a petowner or a petsitter. This can be accessed through their respective main pages. It also implements darkLight, brand, primary, tertiary, secondary colours from Colours in pawsup-frontend/components/styles.js.

[pawsup-frontend/screens/DetailedListing.js:](#)

pawsup-frontend/screens/DetailedListing.js is the frontend screen that allows the user to see a listing that was clicked on in the Services page. The view differs depending on each listing as each listing is unique and has its own traits. This can be accessed through the services page from the petowner's perspective.

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

S:


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/components/Entry.js:

pawsup-frontend/components/Entry.js is the component page that assists the UI of the Services.js screen. This ensures that the UI does not get messed up on the Services page and it represents each Entry on the Services page. It provides the user with a quick glimpse of a listing's details such as title and price and ensures that each entry is clickable so that a user can select a listing for the purpose of either viewing it or booking it.

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`.