


CSCC01

Documentation

Rohan Dey, Ali Orozgani, Mohannad Moustafa Shehata, Vinesh
Benny, Tarushi Thapliyal, Leila Cheraghi Seifabad
19th November, 2021

Table of Contents

Table of Contents	1
Backend	4
backend-database/api/Router.js:	4
backend-database/config/db.js:	44
backend-database/models/User.js:	44
backend-database/models/Listing.js:	44
backend-database/models/Item.js:	45
backend-database/Procfile:	45
backend-database/server.js:	46
Frontend	46
pawsup-frontend/screens/Setting.js:	46
pawsup-frontend/screens/Signup.js:	46
pawsup-frontend/screens/Login.js:	47
pawsup-frontend/screens/PetSitterMain.js:	47
pawsup-frontend/screens/PetOwnerMain.js:	48
pawsup-frontend/screens/AdminMain.js:	48
pawsup-frontend/screens/Services.js:	49
pawsup-frontend/screens/Shop.js:	49
pawsup-frontend/screens/Cart.js:	49
pawsup-frontend/screens/PetSitterModifyListing.js:	49
pawsup-frontend/screens/UpcomingAppointment.js:	50
pawsup-frontend/screens/DetailedListing.js:	50
pawsup-frontend/screens/DetailedItem.js:	50
pawsup-frontend/screens/AdminRemoveUser.js:	50
pawsup-frontend/screens/AdminRemoveListing.js:	50



pawsup-frontend/screens/AdminRemoveProduct.js:	50
pawsup-frontend/screens/AdminAddProduct.js:	50
pawsup-frontend/screens/PreviousStorePurchase.js:	50
pawsup-frontend/screens/PreviousAppointments.js:	50
pawsup-frontend/screens/BookAppointment.js:	51
pawsup-frontend/screens/Checkout.js:	51
pawsup-frontend/components/styles.js:	51
pawsup-frontend/components/KeyboardAvoidingWrapper.js:	52
pawsup-frontend/components/KeyboardAvoidingWrapper2.js:	52
pawsup-frontend/components/Entry.js:	52
pawsup-frontend/components/Entry2.js:	52
pawsup-frontend/components/EntryCart.js:	52
pawsup-frontend/components/Item.js:	53
pawsup-frontend/screens/ListingRating.js:	53
pawsup-frontend/screens/StoreListing.js:	53
pawsup-frontend/navigators/RootStack.js:	53

Backend

Deployment:

The backend itself is deployed on Heroku so that the application does not require a localhost running at all times. This ensures that calls are consistent and there is a lower chance of a server-related failure when ideally used by many users at once.

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.
- `/deleteUser`
 - This is used by the admin in pawsup-frontend/screens/AdminRemoveUser.js to remove listings.
- `/getPreviousOrders`
 - This is used to get the previous orders for a user from the database.
- `/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.

- [/deleteListing](#)
 - This is used by the admin in pawsup-frontend/screens/AdminRemoveProduct.js to remove listings.
- [/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.
- [/sortListings](#)
 - This is used by users in pawsup-frontend/screens/Shop.js to filter store listings by pet type.
- [/addListingRating](#)
 - This is used by users to add a rating for a Listing.
- [/getPreviousBookings](#)
 - This is used by users to get their previous bookings.
- [/makeItem](#)
 - This is used to create items in pawsup-frontend/screens/AdminAddProduct.js.
- [/modifyItem](#)
 - This is used to modify existing items.
- [/getItem](#)
 - This is used to display the items on various Pawsup frontend screens.
- [/addToCart](#)
 - This is used to add items to the cart on various Pawsup frontend screens.
- [/removeFromCart](#)
 - This is used to remove items from the cart in pawsup-frontend/screens/Cart.js.
- [/getInCart](#)
 - This is used to find out what items are in the cart in pawsup-frontend/screens/Cart.js.
- [/getAllItems](#)
 - This is used as a helper to find out what items exist in the database.
- [/deleteItem](#)
 - This is used to remove items from the backend in pawsup-frontend/screens/AdminRemoveProduct.js.
- [/filterPriceItemListings](#)
 - This is used by users in pawsup-frontend/screens/Shop.js to filter store listings by price.

- [/filterPettypeItem Listings](#)
 - This is used by users in pawsup-frontend/screens/Shop.js to filter store listings by pet type.
- [/addItemRating](#)
 - This is used by users to add a rating for an Item in the store.
- [/itemCheckout](#)
 - This is used by users to checkout all the items in their cart.
- [/createPaymentIntent](#)
 - This is used by users in pawsup-frontend/screens/Checkout.js to handle the payment to the Stripe API to make payments.

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", "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>

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 query parameter, where the query 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.
Query Example	email: "email"

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	<p>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.</p>
Body Example	<pre>{</pre>

	<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", "phonenumber": "1010101010", "accounttype": "Petsitter", "pettype": "Dog", "_id": "615fad6612ed1eaae5f942d3", "__v": 0 } }</pre>

Delete a User:

Request

Mimetype	application/json
Method	DELETE
URL	/api/deleteUser
Description	It takes the user email as a query parameter. If the email is empty, or no User exists in the database with that email key, res returns "FAILED". Otherwise, res returns "SUCCESS" and the number of items deleted as data.
Query Example	email: "vineshb100@gmail.com"

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "User deleted successfully", "data": { "deletedCount": 1 } }</pre>

Get a User's previous orders:

Request

Mimetype	application/json
Method	PUT
URL	/api/getPreviousOrders
Description	It takes the user email as a query parameter. If the email is empty, or no User exists in the database with that email key, res returns "FAILED". Otherwise, res returns "SUCCESS" and the user's previous orders as an array as data.
Query Example	email: "vineshb100@gmail.com"

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "User Found Successfully", "data": { "_id": "615f6705842ff29f980bde17", "previousorders": [{ "name": "Dog food", "price": 20, </pre>

	<pre> "image": "https://cdn.discordapp.com/attachments/887519377015078930/90440 3363419017226/21051303084206.png", "quantity": 1, "rating": 1, "_id": "619023638e80f803db561416" }, { "name": "Cat food", "price": 25, "image": "https://media.discordapp.net/attachments/887519377015078930/9044 03374529716244/21051303165405.png?width=468&height=468", "quantity": 1, "rating": 1, "_id": "619023638e80f803db561417" }, { "name": "Dog food", "price": 20, "image": "https://cdn.discordapp.com/attachments/887519377015078930/90440 3363419017226/21051303084206.png", "quantity": 1, "rating": 0.8666666666666666, "_id": "6191e3117aedef6da6c352935" }] } </pre>
--	--

Create a listing for a Petsitter:

Request

Mimetype	application/json
Method	POST
URL	/api/createListing
Description	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.
Body Example	<pre>{ listinowner: "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	It takes in a query parameter, where the query 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.
Query Example	listingowner: “email@t.com”

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	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.
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>

Delete a Listing:

Request

Mimetype	application/json
Method	DELETE
URL	/api/deleteListing
Description	It takes the listing owner email as a query parameter. If the email is empty, or no Listing exists in the database with that listingowner key, res

	returns “FAILED”. Otherwise, res returns “SUCCESS” and the number of items deleted as data.
Query Example	listingowner: “vineshb100@gmail.com”

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Listing deleted successfully", "data": { "deletedCount": 1 } }</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 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.</p>
Body Example	<pre>{ listingowner: “email@t.com”, reason: “BLOCKED”,</pre>

	<pre>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	GET
URL	/api/getPetownerBookings
Description	It takes in a query parameter, where the query 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.
Query Example	petowner: "testuser3@gmail.com"

Response

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

Filter listing by price:

Request

Mimetype	application/json
Method	GET
URL	/api/filterPriceListings
Description	It takes in a query parameter, where the query 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.
Query Example	minprice: 12,

	maxprice: 17
--	--------------

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	<p>It takes in a query parameter, where the query 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.</p>
Query 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>
-----------------------------	---

Sort listing by title, cost, rating, description or feature:

Request

Mimetype	application/json
Method	GET
URL	/api/sortListings
Description	<p>It takes in a query parameter, where the query holds the order in which the Listings should be sorted (ascending or descending) and the sort value that the endpoint should sort listings by.</p> <p>sortListings then sorts all Listings in the database based on these parameters. If sortVal is not "title", "cost", "rating", "description" or "feature", or order is not "asc" or "desc", res returns "FAILED" as its status. Otherwise, res returns "SUCCESS" as its status, and the sorted Listings in an array as data.</p>
Query Example	sortVal: rating, order: asc

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Listings sorted by rating", "data": [{ "_id": "616f12a7328edf63c5478b4b", </pre>

	<pre> "listingowner": "ali@gmail.com", "title": "Ali's Fun Zone", "description": "This is Ali's Fun Zone.", "features": "Support for snakes.", "bookings": [{ "reason": "BLOCKED", "startdate": "2022/02/09", "enddate": "2022/02/10", "_id": "61815d9af9de4cf0fdbf7fe2" }], "__v": 0, "location": "937 Progress Ave, Scarborough, ON M1G 3T8, Canada", "price": 45, "numRatings": 4, "sumRatings": 2, "fullname": "Ali Orozgani", "rating": 0.5 }, { "_id": "6170988edb5491fb2cd57a35", "listingowner": "vineshb100@gmail.com", "title": "Vinesh's Fun Place", "description": "Vinesh's fun region takes care of dogs of all ages! Come have fun with Vinesh by booking this listing!", "location": "1265 Military Trail, Scarborough, ON, M1C1A4", "features": "Complimentary midnight snacks!", "price": 27, "bookings": [{ </pre>
--	--

	<pre> "reason": "BLOCKED", "cost": 10, "startdate": "2021/11/22", "enddate": "2021/11/24", "_id": "6170eb32e3dc905173805e2d" }], "__v": 0, "numRatings": 1, "sumRatings": 1, "fullname": "Vinesh Benny", "rating": 1 }] }</pre>
--	---

Add rating for a Listing:

Request

Mimetype	application/json
Method	PUT
URL	/api/addListingRating
Description	It takes "listingowner" which has the Listing owner's email and a rating from 1-5 in the request body. If "listingowner" is empty or rating is not between 1-5, res returns "FAILED" as its status. Otherwise, the corresponding Listing is found from the listingowner's email and then the rating is added to the database. res then returns SUCCESS as status, and the Listing with the newly updated rating as data
Request Body Example	<pre> { "listingowner": "vineshb100@gmail.com", "rating": 5 }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Listing rating modified successfully", "data": [{ "_id": "6170988edb5491fb2cd57a35", "listingowner": "vineshb100@gmail.com", "title": "Vinesh's Fun Place", "description": "Vinesh's fun region takes care of dogs of all ages! Come have fun with Vinesh by booking this listing!", "location": "1265 Military Trail, Scarborough, ON, M1C1A4", "features": "Complimentary midnight snacks!", "price": 27, "bookings": [{ "reason": "BLOCKED", "cost": 0, "startdate": "2021-11-07", "enddate": "2021-11-07", "paid": false, "_id": "6170e8759ad0878171e61756" }, { "reason": "johnsmith@gmail.com", "cost": 27, "startdate": "2020-11-25", "enddate": "2020-11-25", "paid": true, "_id": "618b0fdcad760e97fac1116" }] }] }</pre>

	<pre> }, { "reason": "johnsmith@gmail.com", "cost": 27, "startdate": "2020-11-23", "enddate": "2020-11-23", "paid": false, "_id": "618b1116cad760e97fad64fb" }, { "reason": "johnsmith1@gmail.com", "cost": 45, "startdate": "2021-11-24", "enddate": "2021-11-24", "paid": false, "_id": "618d5f12b978f18564fb9f7d" }], "__v": 0, "numRatings": 7, "sumRatings": 6.2, "fullName": "Vinesh Benny", "rating": 0.8857142857142858 }]</pre>
--	---

Get Previous Bookings for a Petowner:

Request

Mimetype	application/json
Method	GET
URL	/api/getPreviousBookings
Description	It takes petowner as a query parameter which is email of the petowner. If petowner is empty, res returns "FAILED" as its status. Otherwise, all Listings are searched in the database for bookings of petowner before the current date. If no such bookings are found, res returns "FAILED" as its status. Otherwise res returns "SUCCESS" as its message and an array of all previous bookings as data.
Query Example	petowner: "johnsmith@gmail.com"

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Previous appointments found successfully", "data": [{ "reason": "vineshb100@gmail.com", "cost": 27, "startdate": "2020-11-25", "enddate": "2020-11-25", "paid": true, "_id": "618b0dfcad760e97fac1116", "rating": 0.8857142857142858 }, { "reason": "vineshb100@gmail.com",</pre>

	<pre> "cost": 27, "startdate": "2020-11-23", "enddate": "2020-11-23", "paid": false, "_id": "618b1116cad760e97fad64fb", "rating": 0.8857142857142858 }, { "reason": "ali@gmail.com", "startdate": "2021-11-12", "enddate": "2021-11-13", "_id": "618afb67cad760e97fab1d6e", "rating": 0.3714285714285715 }, { "reason": "ali@gmail.com", "cost": 0, "startdate": "2021-11-12", "enddate": "2021-11-12", "_id": "618b0f22cad760e97fab66ba", "rating": 0.3714285714285715 }, { "reason": "ali@gmail.com", "cost": 0, "startdate": "2021-11-11", "enddate": "2021-11-11", "_id": "618b0f37cad760e97fab8427", "rating": 0.3714285714285715 }, { "reason": "email@t.com", </pre>
--	---

	<pre> "cost": 0, "startdate": "2021-11-12", "enddate": "2021-11-12", "_id": "618b0f22cad760e97fab66ba", "rating": 1.5 }, { "reason": "email@t.com", "cost": 0, "startdate": "2021-11-11", "enddate": "2021-11-11", "_id": "618b0f37cad760e97fab8427", "rating": 1.5 }] } </pre>
--	--

Make new Item:

Request

Mimetype	application/json
Method	POST
URL	/api/makeItem
Description	<p>It takes name, price, description, image, pets and quantity as the request body, where name and description strings, image is a string containing image link for item, pets is an array of strings that signify what pets this item can be used for, and price and quantity are both numbers describing price of 1 item and quantity of available items respectively. If any of those parameters are empty, or if price or quantity are not valid numbers, res returns "FAILED" as its status. Otherwise, once no Item exists in the database with the same name, the new item will be made, and res returns "SUCCESS" as status and the new item as</p>

	data.
Request Body Example	<pre>{ "name": "Testitem", "price": "1.1", "description": "Test", "image": "https://media.discordapp.net/attachments/887519377015078930/904403486379221002/81gJzoCzwZL.png?width=363&height=468", "quantity": "1", "pets": ["cat"] }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Item Creation Successful", "data": { "name": "Testitem", "price": 1.1, "description": "Test", "image": "https://media.discordapp.net/attachments/887519377015078930/904403486379221002/81gJzoCzwZL.png?width=363&height=468", "pets": ["cat"], "quantity": 1, "inCart": [], "_id": "6182b88b8cfa41ce3084a242", "__v": 0 } }</pre>

Modify existing Item:

Request

Mimetype	application/json
Method	PUT

URL	/api/modifyItem
Description	It takes name, price, description, image, pets and quantity as the request body, where name and description strings, image is a string containing image link for item, pets is an array of strings that signify what pets this item can be used for, and price and quantity are both numbers describing price of 1 item and quantity of available items respectively. If any of those parameters are empty, or if price or quantity are not valid numbers, res returns "FAILED" as its status. Otherwise, once an Item exists in the database with the same name, the item will be modified with the new information, and res returns "SUCCESS" as status and the modified item as data.
Request Body Example	<pre>{ "name": "Testitem", "price": "1.90", "description": "Test Modified", "image": "https://media.discordapp.net/attachments/887519377015078930/904403486379221002/81gJzoCzwZL.png?width=363&height=468", "quantity": "10", "pets": ["cat", "dog"] }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Item Modification Successful", "data": { "name": "Testitem", "price": 1.9, "description": "Test Modified", "image": "https://media.discordapp.net/attachments/887519377015078930/904403486379221002/81gJzoCzwZL.png?width=363&height=468", "pets": ["Cat", "Dog"], "quantity": 10, "inCart": [], "_id": "6182b88b8cfa41ce3084a242", "__v": 0 } }</pre>

	<pre>} }</pre>
--	----------------

Get an Item:

Request

Mimetype	application/json
Method	GET
URL	/api/getItem
Description	It takes name as a query parameter. If name is empty, or no Item exists in the database with that name, res returns "FAILED". Otherwise, res returns "SUCCESS" and the found Item as data.
Query Example	name: "dog food"

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Item found", "data": [{ "_id": "617ed4da46f1dacf4be0bb3d", "name": "Dog food", "price": 20, "description": "Science diet - Perfect digestion dog food", "image": "https://cdn.discordapp.com/attachments/887519377015078930/904403363419017226/21051303084206.png", "pets": ["Dog"], "quantity": 861, "__v": 0, "inCart": [{ "user": "vineshb100@gmail.com", "quantity": 7, "_id": "6180bb901cafc54d317ab847"</pre>

	<pre> }] }] }</pre>
--	---

Add an Item to User's Cart:

Request

Mimetype	application/json
Method	PUT
URL	/api/addToCart
Description	It takes name, email and quantity from request body, where name is the name of an Item, email is the email of a User and quantity is the number of the Item to add to cart. If email or name is empty, or quantity is not a valid number, res returns "FAILED" as status. Otherwise, once name is a valid Item name, the item gets added to User's cart and the quantity of the item added to cart is removed from available quantity of item.
Request Body Example	<pre> { "item": "Dog food", "email": "vineshb100@gmail.com", "quantity": "4" }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre> { "status": "SUCCESS", "message": "Update Successful", "data": [{ "_id": "617ed4da46f1dacf4be0bb3d", "name": "Dog food", "price": 20, "description": "Science diet - Perfect digestion dog food", "image":</pre>

	<pre> "https://cdn.discordapp.com/attachments/887519377015078930/90440 3363419017226/21051303084206.png", "pets": ["Dog"], "quantity": 857, "__v": 0, "inCart": [{ "user": "vineshb100@gmail.com", "quantity": 74, "_id": "6180bb901cafc54d317ab847" }] } </pre>
--	--

Remove an Item from User's Cart:

Request

Mimetype	application/json
Method	PUT
URL	/api/removeFromCart
Description	<p>It takes name, email and quantity from the request body, where name is the name of an Item, email is the email of a User and quantity is the number of the Item to remove from cart. If email or name is empty, or quantity is not a valid number, res returns "FAILED" as status. If User does not have Item in the cart, res returns "FAILED" as status.</p> <p>Otherwise, the quantity specified is removed from the User's cart. If the quantity to be removed is more than the quantity in the cart or exactly equal to the quantity in the cart, the item gets removed completely from the User's cart. Otherwise, the item stays in cart but quantity in cart is reduced by quantity stated from request.</p>
Request Body Example	<pre> { "item": "Dog food", "email": "vineshb100@gmail.com", "quantity": "4" } </pre>



	}
--	---

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Update Successful", "data": [{ "_id": "617ed4da46f1dacf4be0bb3d", "name": "Dog food", "price": 20, "description": "Science diet - Perfect digestion dog food", "image": "https://cdn.discordapp.com/attachments/887519377015078930/904403363419017226/21051303084206.png", "pets": ["Dog"], "quantity": 861, "__v": 0, "inCart": [{ "user": "vineshb100@gmail.com", "quantity": 70, "_id": "6180bb901cafc54d317ab847" }] }] }</pre>

Get cart for User:

Request

Mimetype	application/json
Method	GET
URL	/api/getInCart

Description	It takes email as a query parameter. If email is empty, res returns "FAILED". Otherwise, all the items are searched through in database to get all that User has in cart, and res returns "SUCCESS" as its status and the items in User's cart as an array as res' data. res also returns the total price of all the items in the User's cart as totalPrice.
Query Example	email: "vineshb100@gmail.com"

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Items in cart found", "data": [{ "_id": "617ed4da46f1dacf4be0bb3d", "name": "Dog food", "price": 20, "description": "Science diet - Perfect digestion dog food", "image": "https://cdn.discordapp.com/attachments/887519377015078930/904403363419017226/21051303084206.png", "pets": ["Dog"], "quantity": 70, "__v": 0, "inCart": [{ "user": "vineshb100@gmail.com", "quantity": 70, "_id": "6182bf278cfa41ce3084a251" }] }, { "_id": "617ee4b813c7c550929e8ca6", "name": "Cat food", "price": 25, "description": "Science diet - Perfect digestion cat food", "image": "https://media.discordapp.net/attachments/887519377015078930/904403374529716244/21051303165405.png?width=468&height=468", "pets": [</pre>

	<pre> "Cat"], "quantity": 4, "inCart": [{ "user": "vineshb100@gmail.com", "quantity": 4, "_id": "618099d619a955bdd7f4254c" }], "__v": 0 }], "totalPrice": 1500 }</pre>
--	--

Get all Items in database:

Request

Mimetype	application/json
Method	GET
URL	/api/getAllItems
Description	Needs no parameters. Finds all Items in the database. If there are no Items in the database, res returns "FAILED" as its status, otherwise, res returns "SUCCESS" and res returns all the items in database as an array as data.
Query Example	

Response

Mimetype	application/json
Body Example: Individual	<pre> { "status": "SUCCESS", "message": "Items found in database", "data": [{</pre>

	<pre> "_id": "617ed4da46f1dacf4be0bb3d", "name": "Dog food", "price": 20, "description": "Science diet - Perfect digestion dog food", "image": "https://cdn.discordapp.com/attachments/887519377015078930/90440 3363419017226/21051303084206.png", "pets": ["Dog"], "quantity": 861, "__v": 0, "inCart": [{ "user": "vineshb100@gmail.com", "quantity": 70, "_id": "6182bf278cfa41ce3084a251" }] }, { "_id": "617ee4b813c7c550929e8ca6", "name": "Cat food", "price": 25, "description": "Science diet - Perfect digestion cat food", "image": "https://media.discordapp.net/attachments/887519377015078930/9044 03374529716244/21051303165405.png?width=468&height=468", "pets": ["Cat"], "quantity": 668, "inCart": [{ "user": "vineshb100@gmail.com", "quantity": 4, "_id": "618099d619a955bdd7f4254c" }], "__v": 0 }] }</pre>
--	--

Delete an Item:

Request

Mimetype	application/json
Method	DELETE
URL	/api/deleteItem
Description	It takes name as a query parameter. If name is empty, or no Item exists in the database with that name, res returns "FAILED". Otherwise, res returns "SUCCESS" and the number of items deleted as data.
Query Example	name: "Testitem"

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Item deleted successfully", "data": { "deletedCount": 1 } }</pre>

Add rating for an item:

Request

Mimetype	application/json
Method	PUT
URL	/api/addItemRating
Description	It takes "item" which has the Item's name and a rating from 1-5 in the request body. If "item" is empty or rating is not between 1-5, res returns "FAILED" as its status. Otherwise, the corresponding Item is found from the item name and then the rating is added to the database. res then

	returns SUCCESS as status, and the Item with the newly updated rating as data
Request Body Example	<pre>{ "item": "Dog food", "rating": 5 }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Item rating modified successfully", "data": [{ "_id": "617ed4da46f1dacf4be0bb3d", "name": "Dog food", "price": 20, "description": "Science diet - Perfect digestion dog food", "image": "https://cdn.discordapp.com/attachments/887519377015078930/904403363419017226/21051303084206.png", "pets": ["Dog", "Wolf"], "quantity": 861098651, "__v": 0, "inCart": [{ "user": "johnsmith@gmail.com", "quantity": 20, "_id": "61907f8a5a087136d4b5a31d" }], "numRatings": 14, "sumRatings": 12.2, "rating": 0.8714285714285713 }] }</pre>

Checkout Items in a User's cart:

Request

Mimetype	application/json
Method	PUT
URL	/api/itemCheckout
Description	It takes email which has the User's email as the request body. If email is empty, res returns "FAILED" as its status. It then finds all items in the cart for the user. If there are no items in the cart, res returns "FAILED" as its status. Otherwise, all items in the cart are removed from the Item's cart and they are pushed to the previousorders array of the User. res returns "SUCCESS" as its status and returns the User with the updated previousorders array.
Request Body Example	<pre>{ "email": "johnsmith@gmail.com" }</pre>

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "status": "SUCCESS", "message": "Checkout Successful", "data": [{ "_id": "617095b7db5491fb2cd57a2b", "email": "johnsmith@gmail.com", "password": "watermelonarmy", "fullname": "John Smith", "dateofbirth": "1999/10/11", "location": "123 Test Ave, Toronto, ON, M2C4P7", "phonenumber": "4167774444", "accounttype": "Petowner", "pettype": "Watermelon", "__v": 0, "previousorders": [{ "name": "Dog food", </pre>

	<pre> "price": 20, "image": "https://cdn.discordapp.com/attachments/887519377015078930/904403363419017226/21051303084206.png", "quantity": 20, "rating": 0.8714285714285713, "_id": "6197017db563ef6d156aa2af" }, { "name": "Cat food", "price": 25, "image": "https://media.discordapp.net/attachments/887519377015078930/904403374529716244/21051303165405.png?width=468&height=468", "quantity": 186, "rating": 1, "_id": "6197017db563ef6d156aa2b0" }, { "name": "Hamster food", "price": 12, "image": "https://media.discordapp.net/attachments/887519377015078930/904403486379221002/81gJzoCzwZL.png?width=363&height=468", "quantity": 85, "rating": 0.6538461538461539, "_id": "6197017db563ef6d156aa2b1" }] } } </pre>
--	--

Process a Stripe Payment:

Request

Mimetype	application/json
Method	POST
URL	/api/createPaymentIntent
Description	It takes an amount as a query parameter in cents. If the amount is less

	than 0, or the amount is not of the correct format, res returns “FAILED”. Otherwise, res returns “SUCCESS” and the data.
Query Example	name: “Testitem”

Response

Mimetype	application/json
Body Example: Individual	<pre>{ "amount": 25000, "canceledAt": 0, "captureMethod": "Automatic", "clientSecret": "pi_3JxNFbJReyjnby8o1eESs6PE_secret_6tolcJOuy7hjX8bodOjARE6Vn", "confirmationMethod": "Automatic", "created": 907955224, "currency": "cad", "description": null, "id": "pi_3JxNFbJReyjnby8o1eESs6PE", "lastPaymentError": null, "livemode": false, "paymentMethodId": "pm_1JxNFcJReyjnby8oFbn5XGSV", "receiptEmail": null, "shipping": null, "status": "Succeeded", }</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/models/Item.js:

backend-database/models/Item.js defines the Item model with Mongoose Schema to present to the server and MongoDB the format of the Item. It imports Mongoose and each Item model takes in an item name, as a String, price, as a Number, description, as a String, image URL, as a String, array of applicable pets, quantity, as a Number and an inCart array which holds a pet owner and pet sitter's emails along with how many of the product is in their cart. An example of an Item model follows along with how it would be called and utilized:

```
const newItem = new Item({
  name,
  price,
  description,
  image,
  pets,
  quantity,
  inCart
});
```

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 and logout button. 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 and logout button. 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 and logout button. 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/Shop.js:

pawsup-frontend/screens/Shop.js is the frontend screen that provides the petowner and petsitter with the ability to sort and filter through item listings to find the ideal item listing to add to their cart. This handles the visualization of all the store 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 Item.js component file which assists with the UI in terms of columns and formatting.



pawsup-frontend/screens/Cart.js:

pawsup-frontend/screens/Cart.js is the frontend screen that provides the petowner and petsitter with the ability to see what items are in their cart. It also implements various required colours from Colours in pawsup-frontend/components/styles.js along with other styles. Furthermore, it incorporates the EntryCart.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/screens/DetailedItem.js:](#)

`pawsup-frontend/screens/DetailedItem.js` is the frontend screen that allows the user to see an item that was clicked on in the Shop page. The view differs depending on each item as each item is unique and has its own traits. This can be accessed through the shop page from the petowner or petsitter's perspective.

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

`pawsup-frontend/screens/AdminRemoveUser.js` is the frontend screen that provides the admin with the ability to remove a user's account given that they did something that could potentially harm the app's ecosystem. It sends a query to the database, which deletes the account. It also implements various required colours from Colours in `pawsup-frontend/components/styles.js` along with other styles.

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


`pawsup-frontend/screens/AdminRemoveListing.js` is the frontend screen that provides the admin with the ability to remove a pet sitter's Listing given that they did something that could potentially harm the app's ecosystem. It sends a query to the database, which deletes the Listing. It also implements various required colours from Colours in `pawsup-frontend/components/styles.js` along with other styles.

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

`pawsup-frontend/screens/AdminRemoveProduct.js` is the frontend screen that provides the admin with the ability to remove a product from the application's database. It sends a query to the database, which deletes the item. It also implements various required colours from Colours in `pawsup-frontend/components/styles.js` along with other styles.

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

`pawsup-frontend/screens/AdminRemoveProduct.js` is the frontend screen that provides the admin with the ability to add a product to the application's database. It sends a query to the



database, which adds the item with its given traits. It also implements various required colours from Colours in pawsup-frontend/components/styles.js along with other styles.

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

pawsup-frontend/screens/PreviousStorePurchase.js is the frontend screen that provides the user with a list of their previous purchases. It is also a page where the user has the option to rate a given store listing so that Pawsup's owner/admin gets feedback on their products. This is possible with the StoreRating from pawsup-frontend/components/StoreRating.js. The page also queries into the backend to get a user's previous store orders. It also implements various styles from pawsup-frontend/components/styles.js.

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

pawsup-frontend/screens/PreviousAppointments.js is the frontend screen that provides the user with a list of their previous bookings. It is also a page where the user has the option to rate a given listing so that the listing owner gets feedback on their listing and is able to improve it. This is possible with the ListingRating from pawsup-frontend/components/ListingRating.js. The page also queries into the backend to get a user's previous bookings. It also implements various styles from pawsup-frontend/components/styles.js.

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

pawsup-frontend/screens/BookAppointment.js is the frontend screen that provides the user with the ability to book an appointment by entering a start and end date. It sends a query to the database, which checks whether the dates are available and if they are, respective data is sent to the Checkout.js page. It also implements various required colours from Colours in pawsup-frontend/components/styles.js along with other styles.

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

pawsup-frontend/screens/Checkout.js is the frontend screen that allows the user to purchase the booking or item they want given that their credit card goes through the Stripe API once submitted. It sends a query to the database, which fulfills the payment and then books the listing

slot or purchases the item itself. It also implements various required colours from Colours in pawsup-frontend/components/styles.js along with other styles.

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/KeyboardAvoidingWrapper2.js:

pawsup-frontend/components/KeyboardAvoidingWrapper2.js is the component page that implements an epic for the Pawsup app UI. This wrapper without flex 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/components/Entry2.js:](#)

pawsup-frontend/components/Entry2.js is the component page that assists the UI of the UpcomingAppointments.js screen. This ensures that the UI does not get messed up on the Upcoming Appointments page and it represents each Entry on the Upcoming Appointments page. It provides the user with a quick glimpse of a listing's details such as listing owner, start date and end date and ensures that each entry has a cancel button that is clickable so that a user can cancel a booking.

[pawsup-frontend/components/EntryCart.js:](#)


pawsup-frontend/components/EntryCart.js is the component page that assists the UI of the Cart.js screen. This ensures that the UI does not get messed up on the Cart page and it represents each Entry on the Cart page. It provides the user with a quick glimpse of their cart and its details such as title, price and quantity and ensures that each entry is clickable so that a user can remove the item from the cart or change its quantity.

[pawsup-frontend/components/Item.js:](#)

pawsup-frontend/components/Item.js is the component page that assists the UI of the Shop.js screen. This ensures that the UI does not get messed up on the Shop page and it represents each Item on the Shop page. It provides the user with a quick glimpse of an item's details such as name and price and ensures that each entry is clickable so that a user can view it before adding it to their cart.

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

pawsup-frontend/screens/ListingRating.js is the component page that assists the PreviousAppointment.js page, so that a user is able to rate a given listing so that the listing owner gets feedback on their listing and is able to improve it. The page also queries into the backend to add a user's rating to the given listing with the rating applied on the PreviousAppointment.js



page. It also implements AirbnbRating from react-native-ratings, which is the main UI for the ratings component on the PreviousAppointment page.

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

pawsup-frontend/screens/StoreListing.js is the component page that assists the PreviousStorePurchase.js page, so that a user is able to rate a given store listing so that the Pawsup owner/administrator gets feedback on their store listing and is able to improve it. The page also queries into the backend to add a user's rating to the given store listing with the rating applied on the PreviousAppointment.js page. It also implements AirbnbRating from react-native-ratings, which is the main UI for the ratings component on the PreviousAppointment page.

[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.