Software Design Document

For

BirdFood

Prepared by:

Vu Hai Nam

Tran Huynh Nhat Anh

Hoang Danh

Tien Hoang

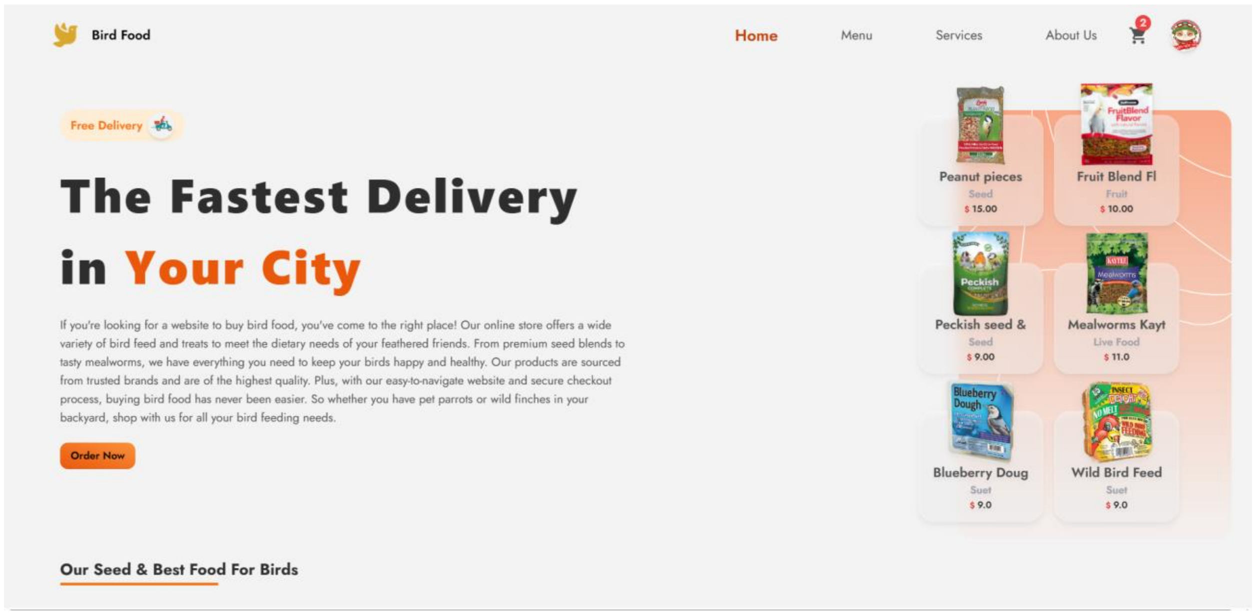
July – 2023

Table of Contents

I. Introduction

BirdFood is an online platform designed to revolutionize the way people purchase bird food and related products. With a seamless user experience and a wide range of quality bird food options, BirdFood aims to cater to bird enthusiasts and pet owners alike.

The purpose of BirdFood is to simplify the process of purchasing bird food and supplies by providing a convenient and user-friendly online marketplace. Through the platform, users can easily browse through a diverse selection of bird food products, compare prices, and make secure purchases.



The Fastest Delivery in Your City

If you're looking for a website to buy bird food, you've come to the right place! Our online store offers a wide variety of bird feed and treats to meet the dietary needs of your feathered friends. From premium seed blends to tasty mealworms, we have everything you need to keep your birds happy and healthy. Our products are sourced from trusted brands and are of the highest quality. Plus, with our easy-to-navigate website and secure checkout process, buying bird food has never been easier. So whether you have pet parrots or wild finches in your backyard, shop with us for all your bird feeding needs.

Our Seed & Best Food For Birds

Key features of BirdFood include:

1. Product Listing and Browsing: BirdFood offers a comprehensive catalog of bird food products, including seeds, pellets, treats, and accessories. Users can explore different categories, filter products based on specific requirements, and view detailed product descriptions.
2. User Authentication: BirdFood provides a secure authentication system that allows users to create accounts, log in, and manage their profiles. Registered users can benefit from features like order history tracking, personalized recommendations, and saved payment methods.
3. Shopping Cart Management: BirdFood enables users to add products to their shopping carts, review their selections, adjust quantities, and remove items if needed. The shopping

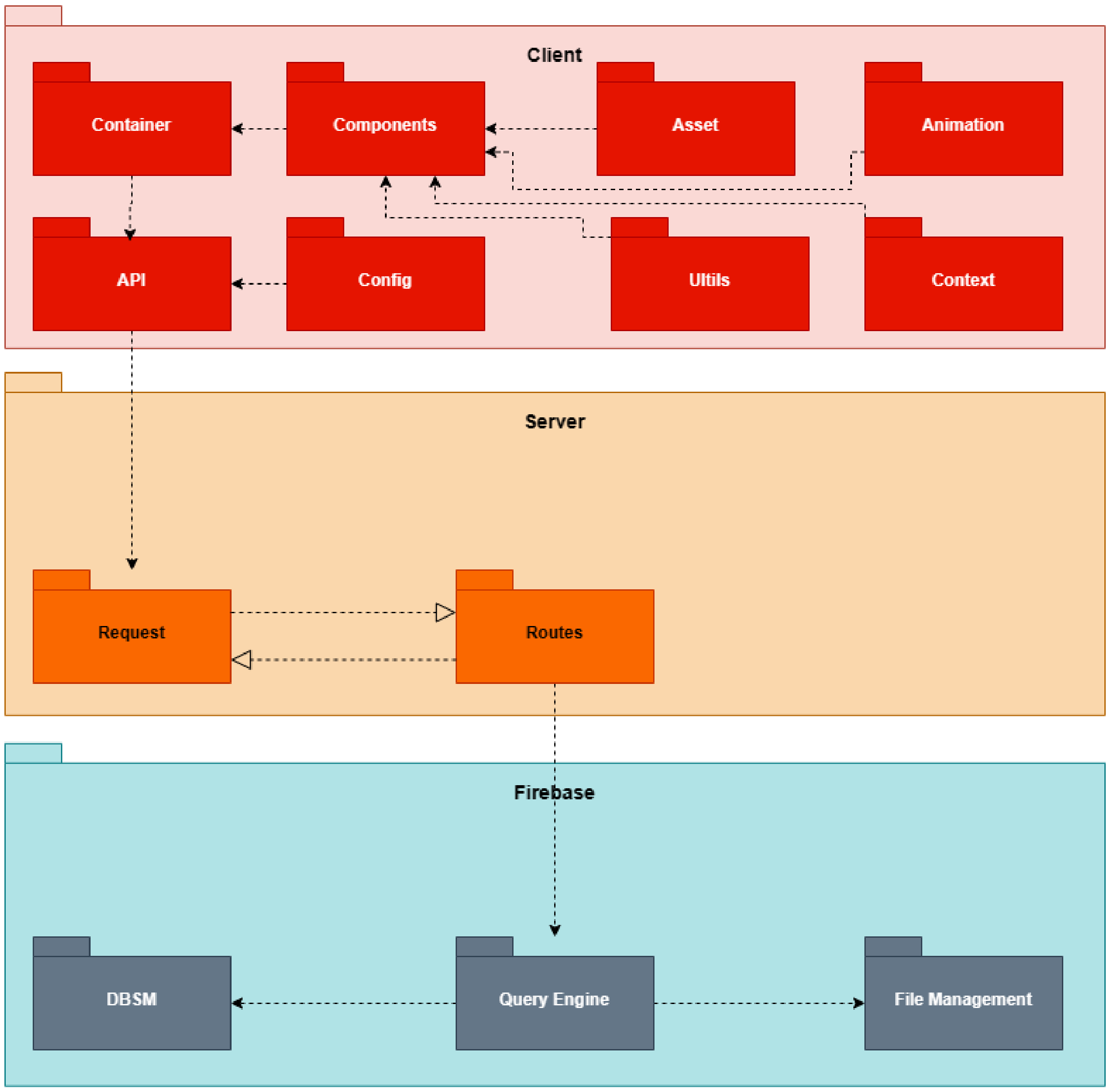
cart provides a seamless checkout experience by calculating totals, applying discounts or promotions, and displaying the estimated delivery time.

4. Order Placement: Once users are satisfied with their selections, BirdFood allows them to place orders effortlessly. Users can review their order details, choose a preferred shipping method, and provide shipping information. The system generates order confirmation emails and provides real-time order tracking for customers.
5. Payment Integration with Stripe: BirdFood integrates with the Stripe payment gateway, ensuring secure and seamless payment processing. Users can safely enter their payment information, choose from various payment methods, and receive confirmation of successful transactions. Stripe's robust security measures help safeguard sensitive customer data.
6. Account Management: BirdFood enables users to manage their accounts, including updating personal information, managing payment methods, and adjusting notification preferences. This feature provides a personalized experience and allows users to maintain control over their BirdFood interactions.
7. By offering a user-friendly interface, a vast selection of products, secure transactions, and convenient features, BirdFood aims to become the go-to platform for bird lovers and pet owners seeking high-quality bird food and supplies.

By offering a user-friendly interface, a vast selection of products, secure transactions, and convenient features, BirdFood aims to become the go-to platform for bird lovers and pet owners seeking high-quality bird food and supplies.

II. Overview

1. Code Packages/ Namespaces



Package descriptions & package class naming conventions

No	Package	Description
01	Context	The "context" package contains the core components for managing the application state using a state management library such as Redux. It includes actions and reducers.
02	Animation	The "animation" package contains reusable animations and transition effects that can be used to enhance the user interface of the application. It provides various animation presets and configurations for creating visually appealing and interactive elements.
03	Assets	The “assets” package contains various static assets used in the application, such as icons, CSS files, and images.
04	Ultils	The "utils" package typically contains utility functions, constants, or reusable code snippets that provide helper functionalities. In this case, it includes status styles and sample data.
05	Component	The “Component” package contains reusable UI components that are used to build the user interface of the application.
06	Containers	The "Containers" package typically includes higher-level components or containers that handle data fetching and management, and provide the logic and state for the presentation components.
07	Config	The “Config” package contains configuration files and settings that are used to set up and customize the behavior of the application.
08	API	The "API" package contains modules or files responsible for interacting with external APIs or services in the application.
09	Request	The "Request" package configures the HTTP server and handles HTTP requests and responses in the application.
10	Routes	The "Routes" package handles the routing and request handling within the application, defining endpoints and their logic.

2. Database Schema

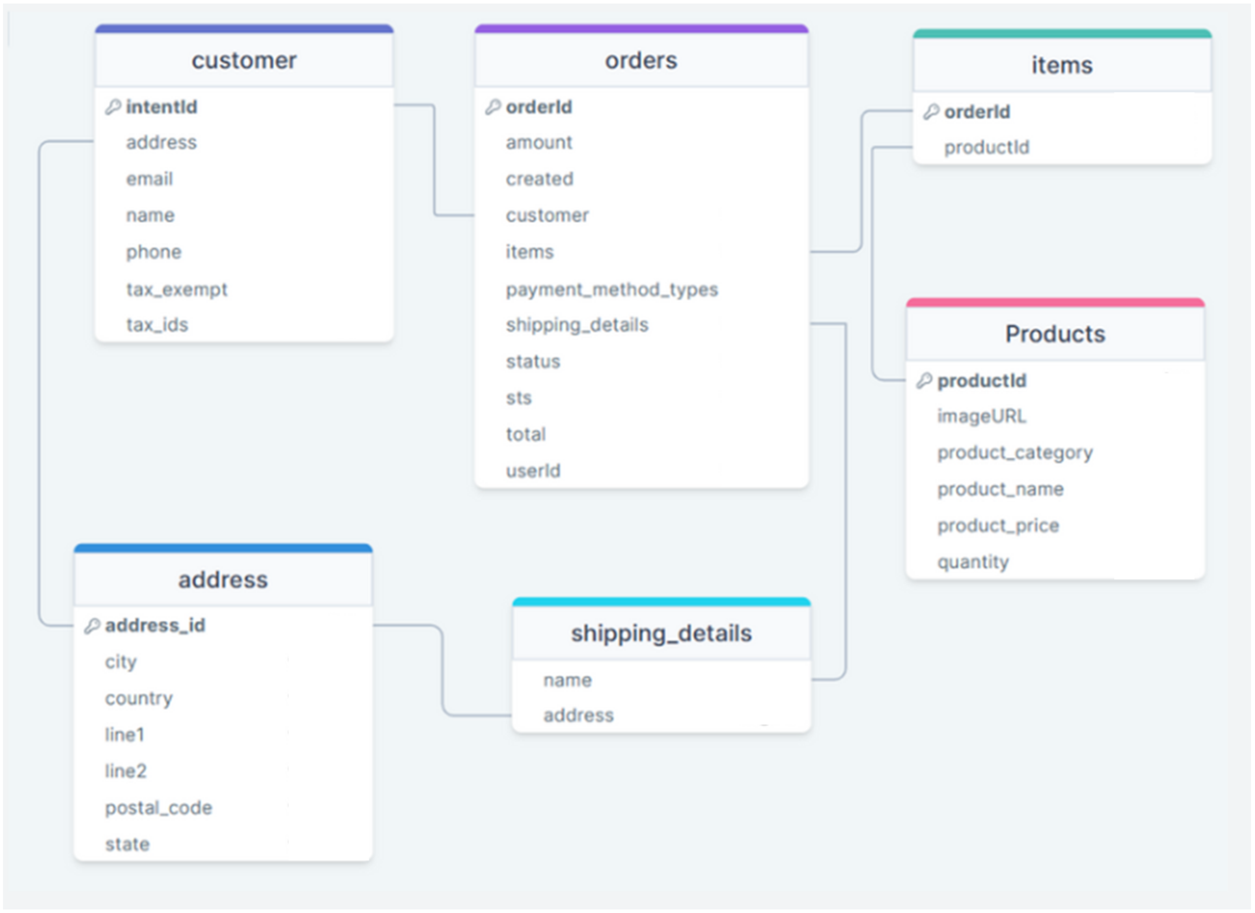
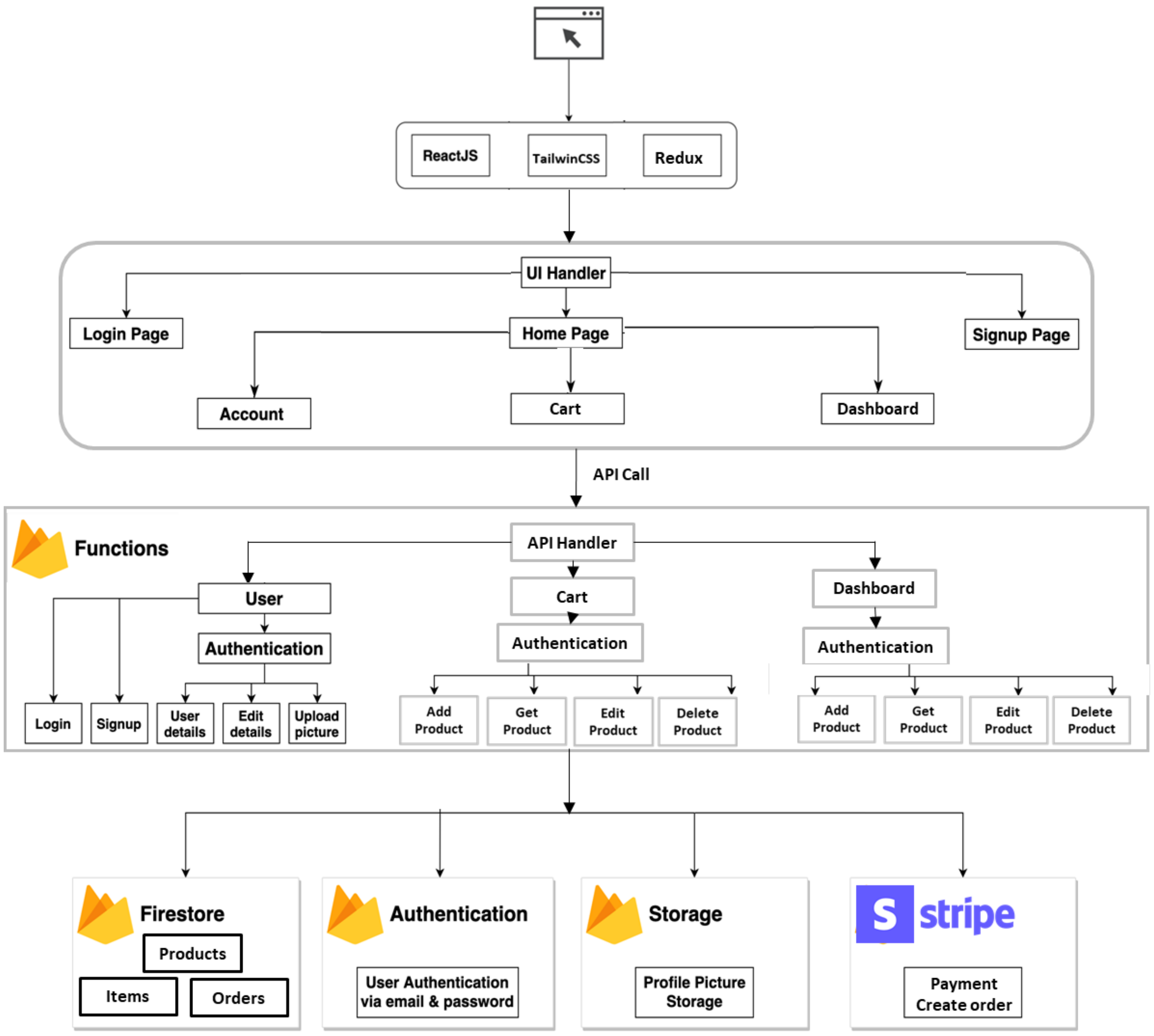


Table descriptions & package class naming conventions

No	Table	Description
01	Customer	The "Customer" table stores information about customers or users in the application such as intent Id, address, email, name, phone, tax exempt, tax ids - Primary keys: intenId - Foreign keys: customer
02	Orders	The "Order" table stores information about orders placed by customers in the application such as order id, amountm created, customer, items, payment method types, shipping details, status, sts, total, userID - Primary keys: orderID - Foreign keys: orderID
03	Items	The "Items" table stores information about individual items or products available in the application such as order id, product id - Primary keys: orderID - Foreign keys: items, productid
04	Products	The "Products" table stores information about the products available in the application such as product id, image URL, product category, product name, product price, quantity
05	Shipping Details	The "ShippingDetails" table stores information about the shipping details associated with customer orders such as name, address - Primary keys: No primary key - Foreign keys: addressid, shipping_details
06	Address	The "Address" table stores information about addresses associated with customers or users in the application such as addressid , city, country, line 1, line 2, postercode, state - Primary keys: addressId - Foreign keys: address

III. Code design

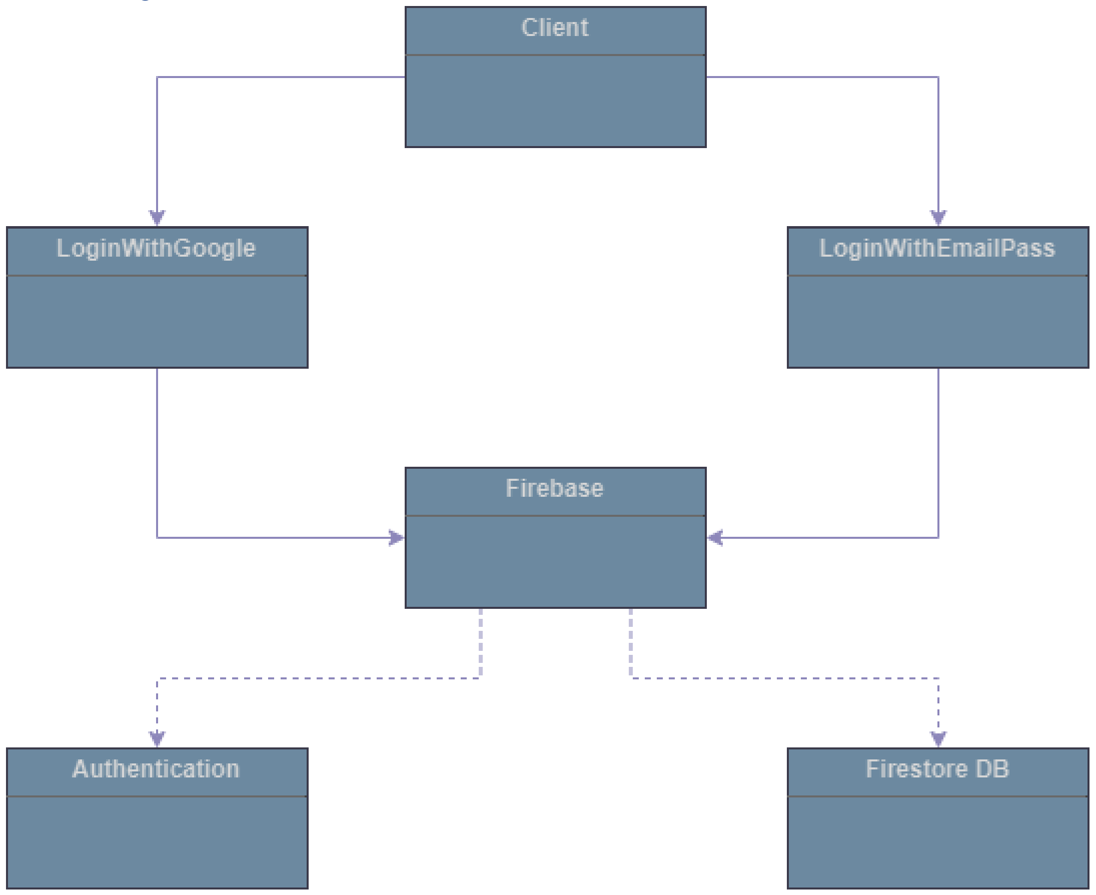
1. Overview



2. Client

2.1 Login

a. Class Diagram



b. Class Specifications

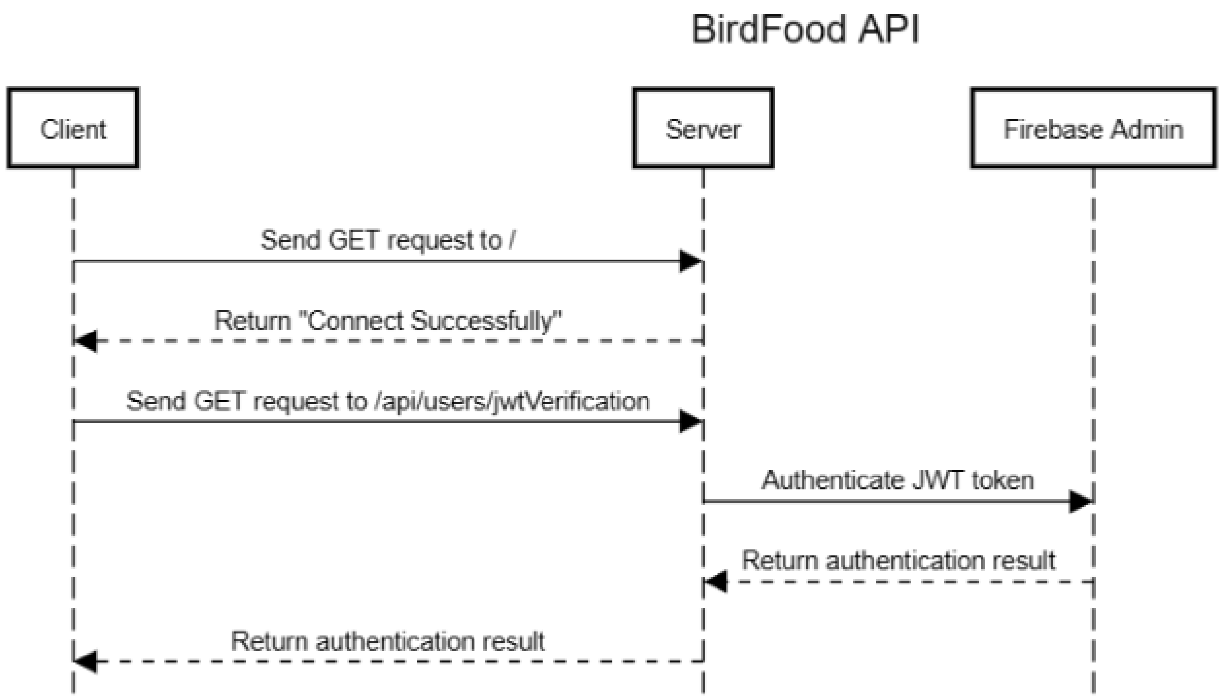
Login.jsx

No	Funtion	Description
01	useState	The "useState" hook from React is used to manage state variables within the component. In this case, it is used to manage the state of user email, password, confirm password, and sign-up status (isSignUp).
02	useEffect	The "useEffect" hook is used to perform side effects in the component. In this case, it checks if the user is already authenticated and navigates to the home page if the user is logged in.
03	loginWithGoogle	This function handles the login with Google functionality. It uses the Firebase authentication API to authenticate the user with their Google account. Upon successful authentication, the user's details

		are obtained, and a JWT token is validated. The user details are then set in the Redux store, and the user is redirected to the home page.
04	signUpWithEmailPass	This function handles the sign-up with email and password functionality. It checks if the required fields (user email, password, confirm password) are not empty. If the passwords match, the user is created with the provided email and password. The user's details are then set in the Redux store, and the user is redirected to the home page.
05	signInWithEmailPass	This function handles the sign-in with email and password functionality. It checks if the user email and password are not empty. If the provided credentials are valid, the user is authenticated using the Firebase authentication API. The user's details are then set in the Redux store, and the user is redirected to the home page.
06	useDispatch	The "useDispatch" hook from React Redux is used to dispatch actions to the Redux store. It is used to dispatch actions related to setting user details and displaying alerts
07	useSelector	The "useSelector" hook from React Redux is used to access data from the Redux store. In this case, it is used to access the user and alert state variables from the store.
08	motion	The “motion” object from the Framer Motion library is used to apply animations to certain elements in the component.
09	react-router-dom	The “useNavigate” hook from the “react-router-dom” package is used to programmatically navigate to different routes in the application.
10	firebase/auth	The functions from the Firebase authentication API are imported to handle user authentication operations such as signing in with email and password, signing up, and signing in with Google.
11	firebase.config	The "app" object from the Firebase configuration file is imported to initialize the Firebase app instance.
12	api	The "validateUserJWTToken" function from the "api" module is imported to validate the JWT token obtained during user authentication.
13	context/actions	The "setUserDetails" action from the "userActions" module and the "alertInfo" and "alertWarning" actions from the "alertActions"

		module are imported to dispatch actions for setting user details and displaying alerts respectively.
14	aleart	The “alert” variable is used to access the alert state variable from the Redux store
15	user	The “user” variable is used to access the user state variable from the Redux store.
16	LoginInput	The "LoginInput" component is imported from the "components" module. It represents an input field for the login form.

c. Sequence Diagram(s)


BirdFood API
Client
Server
Firebase Admin
Send GET request to /
Return "Connect Successfully"
Send GET request to /api/users/jwtVerification
Authenticate JWT token
Return authentication result
Return authentication result

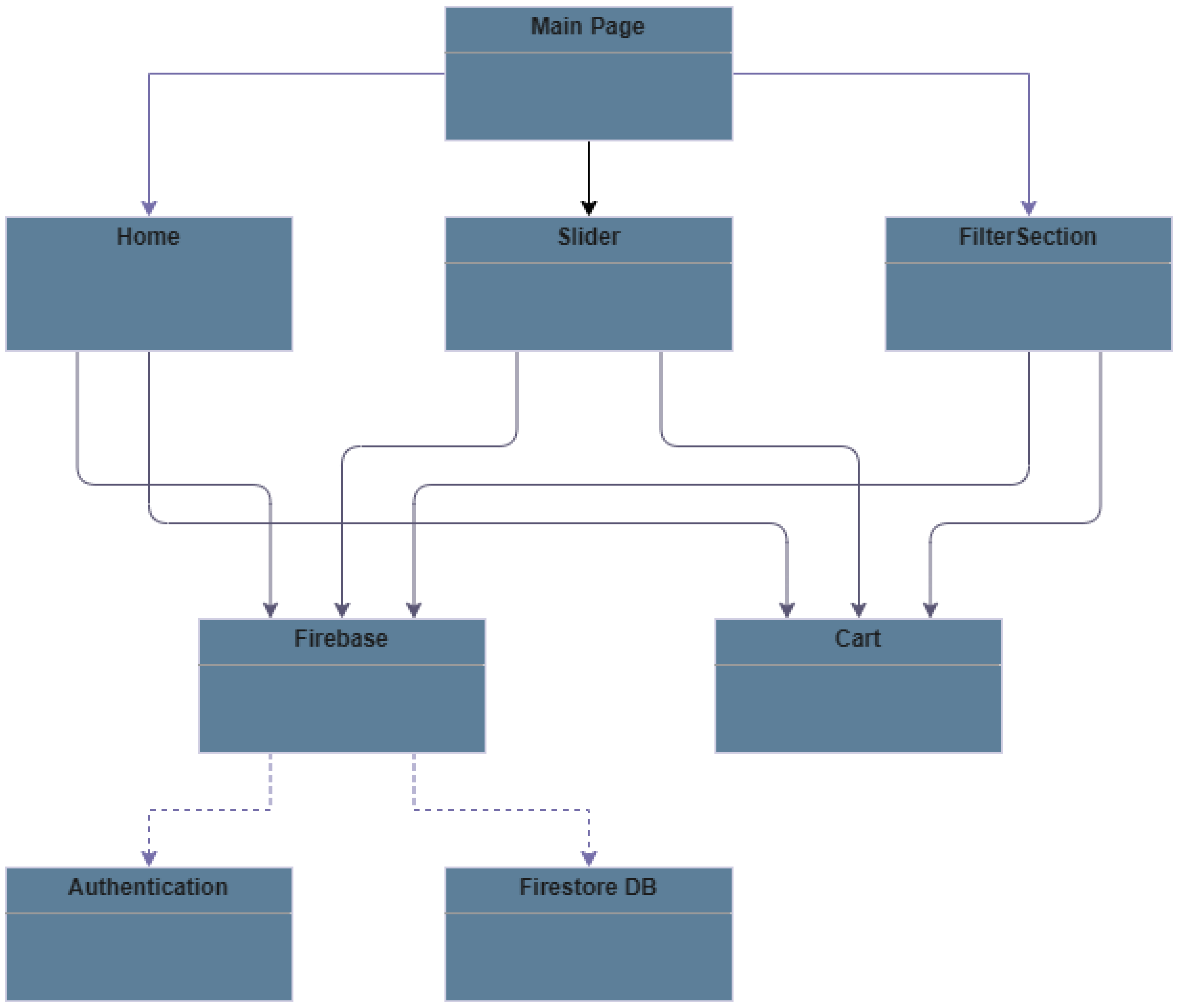
d. API queries

No	Funtion	Description
01	Router	The "Router" function from the Express framework is used to create a new router instance. It allows defining routes and handling HTTP requests for specific routes.
02	admin	The "admin" object is imported from the "firebase-admin" module. It is used to access Firebase Admin SDK functionalities, such as verifying JWT tokens.
03	data	The "data" variable is initialized as an empty array. It is used to store data fetched from the database or other sources.

04	router.get("/jwtVerfication")	This function defines a GET route for the "/jwtVerification" path. It handles the verification of a JSON Web Token (JWT) sent in the "Authorization" header. If the token is valid, it returns a JSON response with the decoded value. If the token is not found or invalid, it returns an appropriate error response.
05	req.headers.authorization	The "Authorization" header of the request is accessed using "req.headers.authorization".
06	Token	The "token" variable is assigned the value of the JWT extracted from the "Authorization" header.
07	admin.auth().verifyIdToken(token)	The "verifyIdToken" function from the Firebase Admin SDK is used to verify the authenticity and integrity of the JWT token. If the token is valid, it returns the decoded value. If the token is invalid, it throws an error.
08	return res.status().send()	These statements are used to send different HTTP responses based on the verification result. If the token is not found, it sends a 500 status with the message "Token Not Found". If the token is invalid, it sends a 500 status with the message "Unauthorized access". If the token is valid, it sends a 200 status with the success flag and the decoded value in the response body.
09	catch(err)	This block of code is used to catch any errors that occur during the token verification process. It sends a response with a success flag set to false and an error message.

2.2 Home Page

a. Class Diagram



b. Class Specfication

Main.jsx

No	Funtion	Description
01	Main	The "Main" function component represents the main page of the application. It includes the header, home section, home slider, and filter section. It also conditionally renders the cart component based on the "isCart" state.
02	useDispatch	The "useDispatch" hook from React Redux is used to dispatch actions to the Redux store. It is used to dispatch the "setAllProducts" action.

03	useSelector	The "useSelector" hook from React Redux is used to access data from the Redux store. It is used to retrieve the "products" and "isCart" state variables from the store.
04	useEffect	The "useEffect" hook is used to perform side effects in the component. In this case, it checks if the "products" state variable is not available and retrieves all products using the "getAllProducts" API. Once the data is fetched, it dispatches the "setAllProducts" action to store the data in the Redux store.
05	getAllProducts	The "getAllProducts" function from the "api" module is imported to retrieve all products from the API.
06	setAllProducts	The "setAllProducts" action from the "productActions" module is imported to dispatch an action to set all products in the Redux store.
07	return	The return statement renders the main content of the page. It includes the header, home section, home slider, and filter section. Additionally, the cart component is conditionally rendered if the "isCart" state is true.

Home.jsx

No	Funtion	Description
01	Home	The "Home" function component represents the home section of the application. It includes information about free delivery, a description of the website, and a list of random products.
02	motion	The "motion" object is imported from the "framer-motion" library. It provides animation functionalities for components.
03	buttonClick	The "buttonClick" animation is imported from the "animations" module. It represents the animation effect for buttons.
04	staggerFadeInOut	The "staggerFadeInOut" animation is imported from the "animations" module. It represents the staggered fade-in and fade-out animation effect for the list of random products.
05	Delivery	The "Delivery" image is imported from the "assets" module. It represents the image of a delivery truck.
06	HeroBg	The "HeroBg" image is imported from the "assets" module. It represents the background image for the home section.
07	randomData	The "randomData" array is imported from the "utils/styles" module. It contains sample data for the random products displayed in the component.
08	return	The return statement renders the content of the home section. It includes a section for free delivery, a heading, a description, and a button to order. It also includes a section with an image and a list of random products. The

		products are mapped from the "randomData" array and animated using the "staggerFadeInOut" animation.

Home Slider

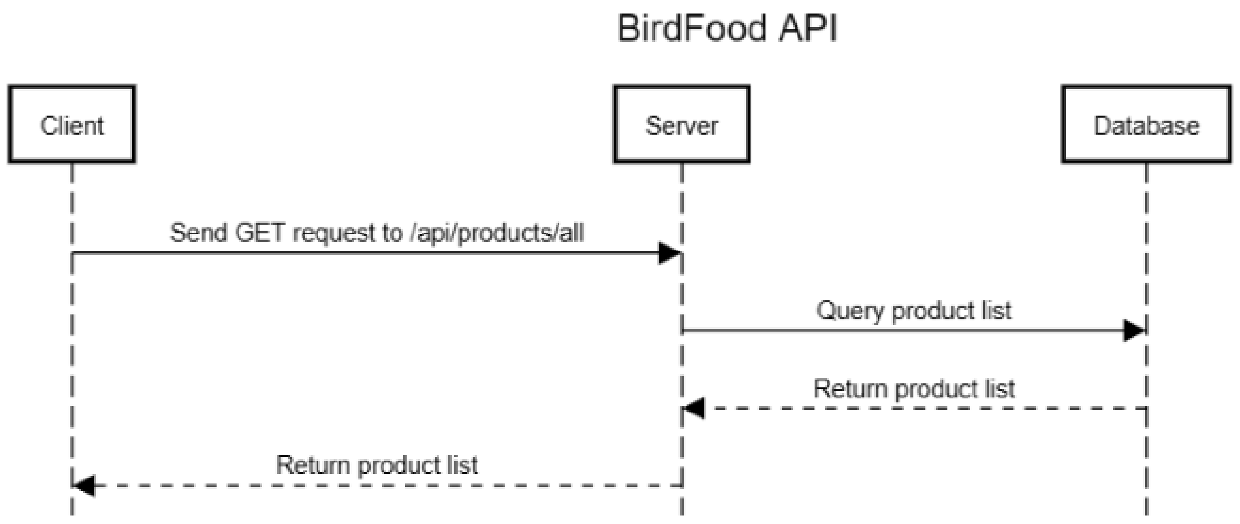
No	Function	Description
01	HomeSLider	The "HomeSLider" function component represents the slider section of the home page. It includes a heading and a slider component.
02	motion	The "motion" object is imported from the "framer-motion" library. It provides animation functionalities for components.
03	Slider	The "Slider" component is imported from the "../components" module. It represents the slider component that displays images or content in a slideshow format.
04	return	The return statement renders the content of the slider section. It includes a heading and the "Slider" component. The heading represents the title of the slider section and the "Slider" component represents the actual slider content.

FilterSection

No	Function	Description
01	FilterSection	The "FilterSection" function component represents the filter section of the home page. It includes a heading, a list of filter cards, and a list of slider cards based on the selected category.
02	useState	The "useState" hook is imported from the "react" library. It allows the component to manage stateful values.
03	useSelector	The "useSelector" hook is imported from the "react-redux" library. It allows the component to access the selected state from the Redux store.
04	staggerFadeInOut	The "staggerFadeInOut" animation is imported from the "animations" module. It represents the staggered fade-in and fade-out animation effect for the filter cards.
05	statuses	The "statuses" array is imported from the "../utils/styles" module. It contains the list of filter categories.
06	SliderCard	The "SliderCard" component is imported from the "./SliderCard" module. It represents the card component used in the slider.
07	return	The return statement renders the content of the filter section. It includes a heading, a list of filter cards, and a list of slider cards. The filter cards are mapped from the "statuses" array and animated using the "staggerFadeInOut" animation. The slider cards are filtered and mapped from the "products" state based on the selected category.

08	FilterCard	The "FilterCard" component represents an individual filter card within the filter section. It receives props such as data, index, category, and setCategory. It is used by the "FilterSection" component to render the filter cards.
09	key	The "index" prop is used as the key for each filter card when mapping over the "statuses" array.
10	onClick	The "onClick" event handler is used to set the selected category when a filter card is clicked.

c. Sequence Diagram(s)

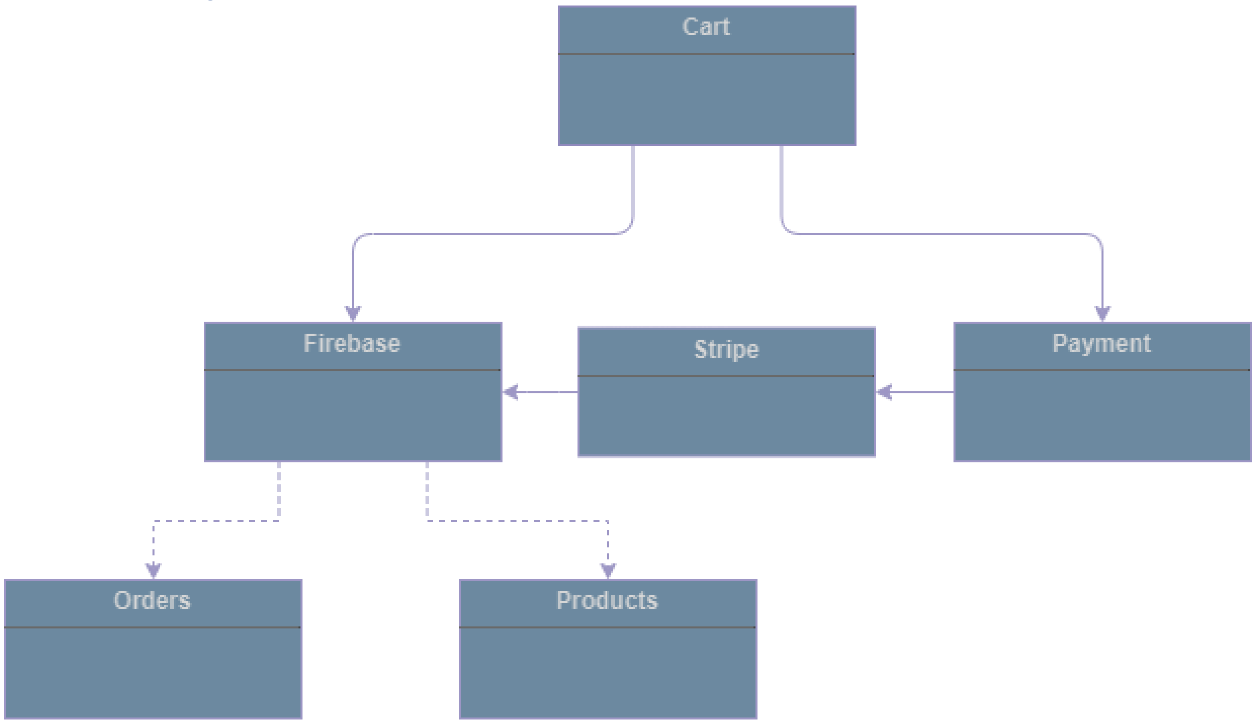


d. API queries

No	Function	Description
01	getAllProducts - Description: This function retrieves all products from the "products" collection in the database. - Dependencies: db (assumed to be the database connection object) - Method: GET - Endpoint: "/all" - Return: JSON response with the fetched products data or an error message.	Function to fetch all products from the database.

2.3 Cart

a. Class Diagram



b. Class Specfication

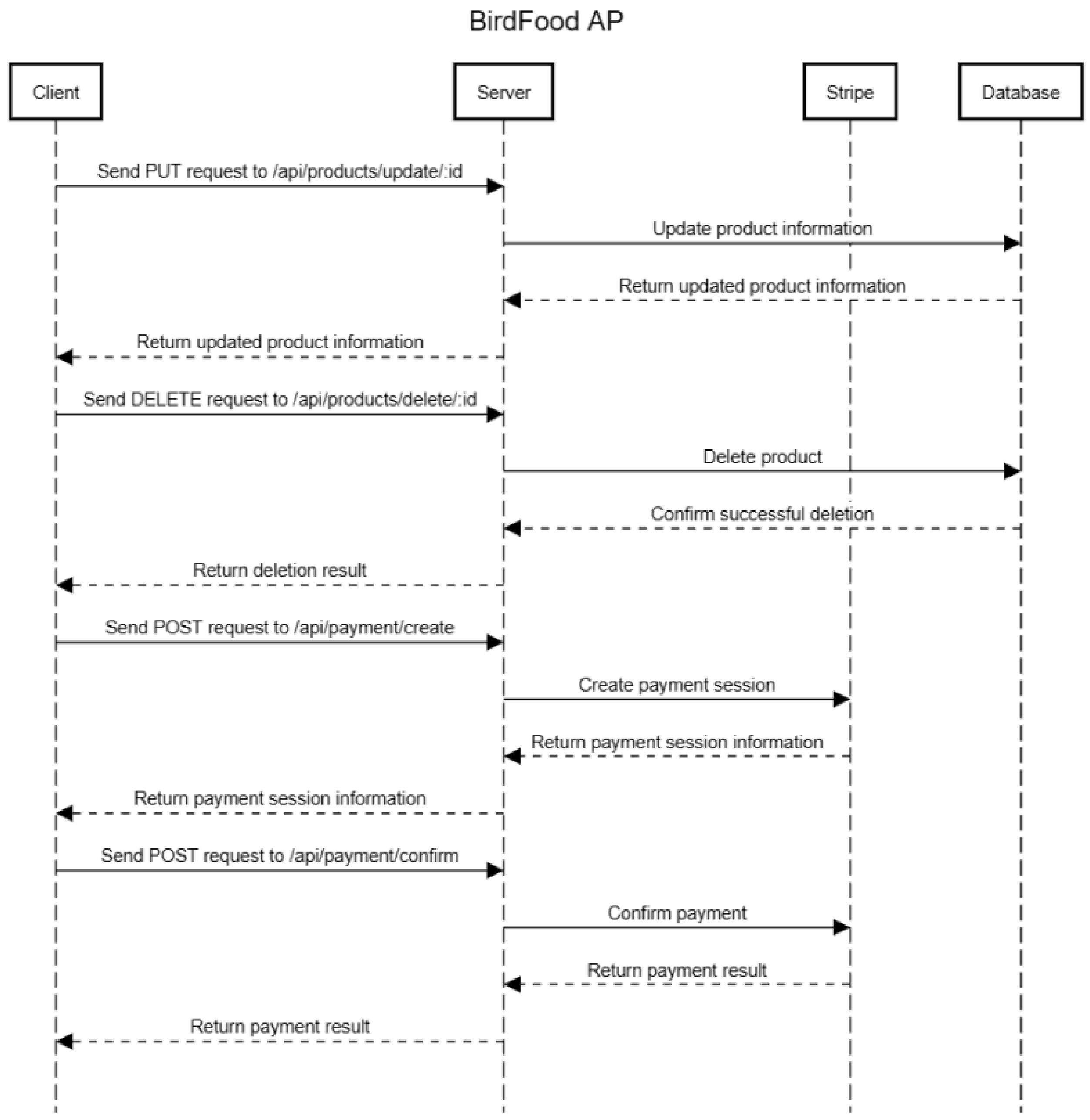
Cart.jsx

No	Function	Description
01	Cart	The "Cart" function component represents the cart section of the application. It displays the items in the cart, calculates the total price, and provides a checkout button.
	motion	The "motion" object is imported from the "framer-motion" library. It provides animation functionalities for components.
	useEffect	The "useEffect" hook is imported from the "react" library. It allows the component to perform side effects when the component is rendered.
	useState	The "useState" hook is imported from the "react" library. It allows the component to manage stateful values.
	useDispatch	The "useDispatch" hook is imported from the "react-redux" library. It allows the component to dispatch actions to the Redux store.
	useSelector	The "useSelector" hook is imported from the "react-redux" library. It allows the component to access the selected state from the Redux store.
	buttonClick	The "buttonClick" animation is imported from the "animations" module. It represents the button click animation effect.
	slideIn	The "slideIn" animation is imported from the "animations" module. It represents the slide-in animation effect for the cart section.
	BiChevronsRight	The "BiChevronsRight" icon is imported from the "../assets/icons" module. It represents the icon for closing the cart section.

	FcClearFilters	The "FcClearFilters" icon is imported from the "../assets/icons" module. It represents the icon for clearing the filters.
	MdAttachMoney	The "MdAttachMoney" icon is imported from the "../assets/icons" module. It represents the icon for currency.
	alertNULL	The "alertNULL" action is imported from the "../context/actions/alertActions" module. It represents the action for clearing the alert message.
	alertSuccess	The "alertSuccess" action is imported from the "../context/actions/alertActions" module. It represents the action for displaying a success alert message.
	setCartOff	The "setCartOff" action is imported from the "../context/actions/displayCartAction" module. It represents the action for hiding the cart section.
	baseURL	The "baseURL" variable is imported from the "../api" module. It represents the base URL for API requests.
	getAllCartItems	The "getAllCartItems" function is imported from the "../api" module. It fetches all the items in the cart from the server.
	increaseItemQuantity	The "increaseItemQuantity" function is imported from the "../api" module. It increases the quantity of an item in the cart.
	setCartItems	The "setCartItems" action is imported from the "../context/actions/cartAction" module. It represents the action for updating the cart items in the Redux store.
	handleCheckOut	The "handleCheckOut" function is called when the user clicks the checkout button. It sends a POST request to create a checkout session and redirects the user to the checkout page.
	return	The return statement renders the content of the cart section. It includes the cart items, total price, and checkout button. The cart items are mapped from the "cart" state and animated using the "staggerFadeInOut" animation.
02	CartItemCard	The "CartItemCard" component represents an individual item card within the cart section. It receives props such as index and data. It is used by the "Cart" component to render the item cards.
	incrementCart	The "incrementCart" function is called when the user clicks the increment button for a cart item. It increases the quantity of the item and updates the cart items in the Redux store.
	decrementCart	The "decrementCart" function is called when the user clicks the decrement button for a cart item. It decreases the quantity of the item and updates the cart items in the Redux store.
	alertSuccess	The "alertSuccess" action is imported from the "../context/actions/alertActions" module. It represents the action for displaying a success alert message.
	setCartItems	The "setCartItems" action is imported from the "../context/actions/cartAction" module. It represents the action for updating the cart items in the Redux store.

	alertNULL	The "alertNULL" action is imported from the "../context/actions/alertActions" module. It represents the action for clearing the alert message.
	return	The return statement renders the content of the item card. It displays the item details, quantity, and total price. The quantity can be incremented or decremented using the respective buttons.
	key	The "index" prop is used as the key for each item card when mapping over the "cart" array.
	incrementCart	The "incrementCart" function is called when the increment button is clicked. It dispatches the "alertSuccess" action, increases the quantity of the cart item, fetches the updated cart items from the server using the "increaseItemQuantity" function, and updates the cart items in the Redux store using the "setCartItems" action.
	decrementCart	The "decrementCart" function is called when the decrement button is clicked. It dispatches the "alertSuccess" action, decreases the quantity of the cart item, fetches the updated cart items from the server using the "increaseItemQuantity" function, and updates the cart items in the Redux store using the "setCartItems" action.
	useEffect	The "useEffect" hook is used to update the total price of the cart item when the quantity or cart items change.
	setItemTotal	The "setItemTotal" function is used to update the total price of the cart item in the state.
03	export default	The "Cart" component is exported as the default export of the module.

c. *Sequence Diagram(s)*


BirdFood AP
Client
Server
Stripe
Database
Send PUT request to /api/products/update/:id
Update product information
Return updated product information
Return updated product information
Send DELETE request to /api/products/delete/:id
Delete product
Confirm successful deletion
Return deletion result
Send POST request to /api/payment/create
Create payment session
Return payment session information
Return payment session information
Send POST request to /api/payment/confirm
Confirm payment
Return payment result
Return payment result

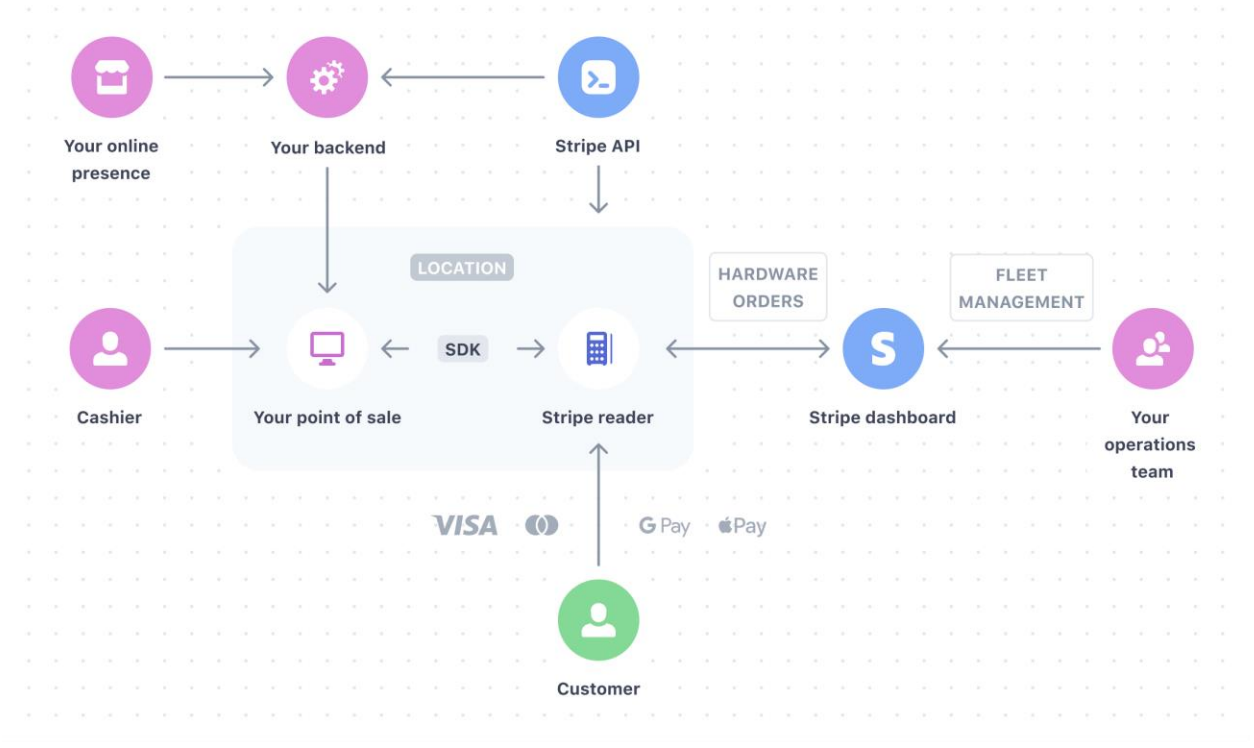
d. *API queries*

No	Method	Description

01	`POST /addToCart/:userId`	This method is used to add a product to the user's cart. It requires the `userId` parameter in the URL and expects the `productId` of the product to be added in the request body. If the product already exists in the cart, the quantity is increased by 1. If the product does not exist in the cart, a new cart item is created with the provided product details (such as `product_name`, `product_category`, `product_price`, and `imageURL`). The response includes the success status and the updated cart item.
02	`POST /updateCart/:user_id`	This method is used to update the quantity of a cart item. It requires the `user_id` parameter in the URL and expects the `productId` and `type` (either "increment" or "decrement") as query parameters. If the `type` is "increment", the quantity of the cart item is increased by 1. If the `type` is "decrement", the quantity is decreased by 1. If the quantity reaches 0 after decrementing, the cart item is deleted from the cart. The response includes the success status and the updated cart item or the deletion result.
03	`GET /getCartItems/:user_id`	This method is used to retrieve all the cart items for a specific user. It requires the `user_id` parameter in the URL. The method retrieves all the cart items associated with the user from the database and returns them as an array. The response includes the success status and the array of cart items.

2.4 Payment (Stripe)

a. Class Diagram



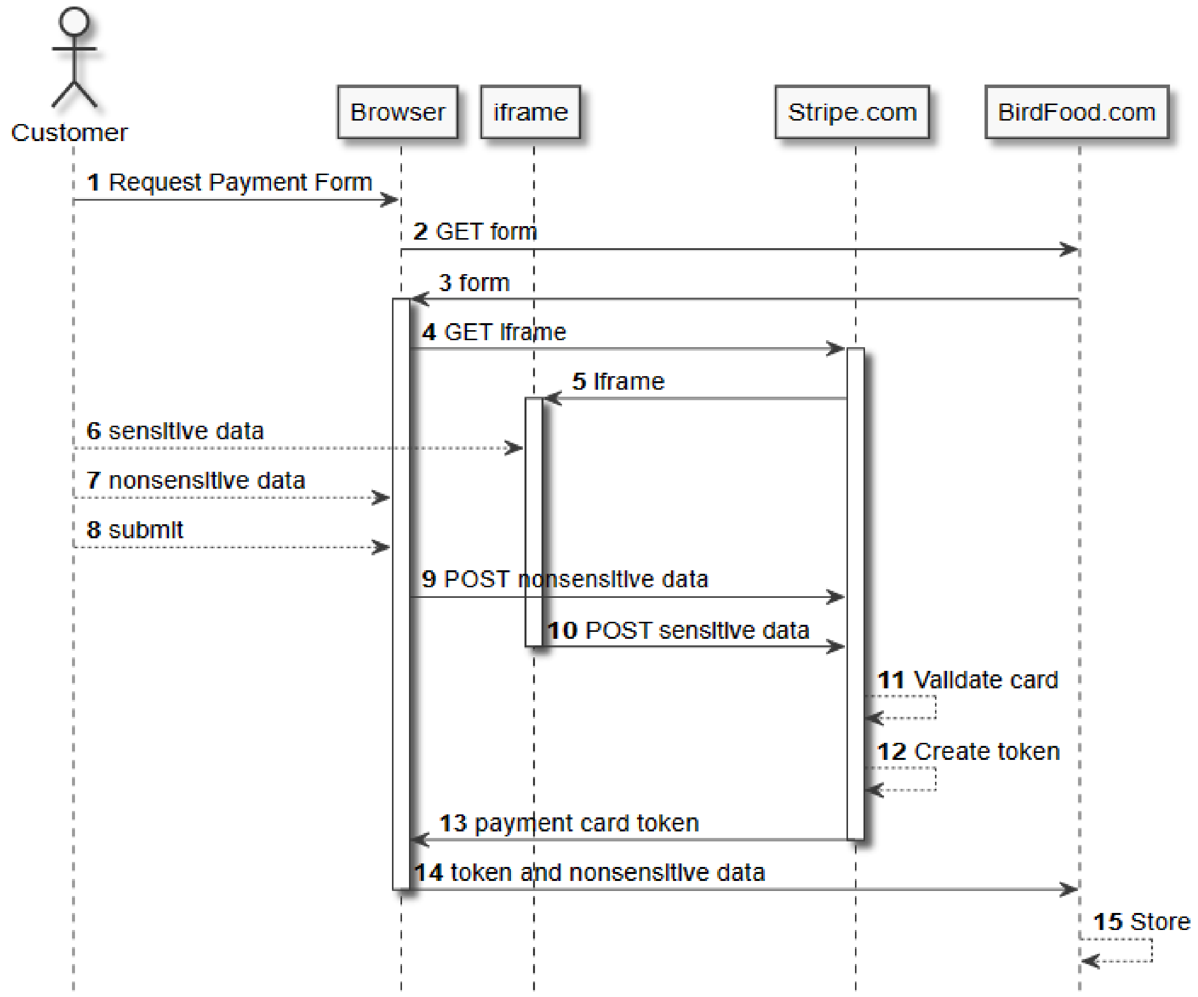
b. Scope of integration

The full scope of an integration consists of four major steps.

1. Use the sample integration to get up and running with an integration quickly.
2. Design your integration to create in-person payments.
3. Integrate the SDK in your application. Use the simulated reader to emulate reader behavior for all the Terminal flows while building your initial integration.
4. Order a physical reader and test card.

From there, explore the docs to see all you can do with your Terminal integration. We recommend testing your integration and reviewing the checklist before going live.

c. Sequence Diagram(s)



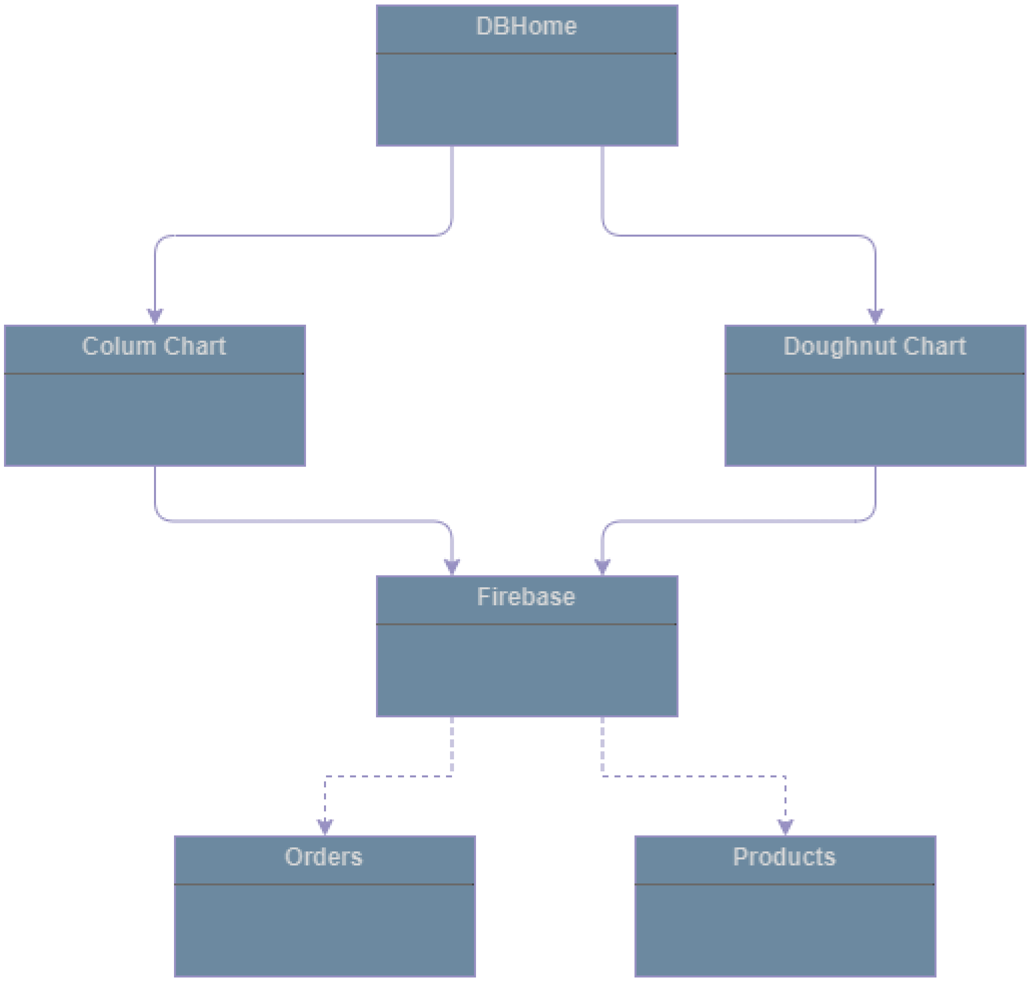
d. API queries

No	Method	Description
01	`POST /create-checkout-session`	This method is used to create a new checkout session for the Stripe payment. It expects the user's information, cart items, and total amount in the request body. The method first creates a customer in Stripe using the `customer.create` method, storing the user's ID, cart, and total amount in the customer's metadata. Then, it constructs an array of line items based on the cart items, including the product details and quantity. Finally, it creates a checkout session using the `checkout.sessions.create` method with the line items, customer ID,

		and other relevant information. The response includes the URL of the checkout session.
02	`POST /webhook`	This method is used to handle webhook events from Stripe. It receives the webhook payload and signature in the request. If an `endpointSecret` is provided, it verifies the signature and constructs the event using the `stripe.webhooks.constructEvent` method. Otherwise, it directly retrieves the event data from the request body. The method handles the `checkout.session.completed` event by retrieving the customer details and creating an order using the `createOrder` function. It responds with a 200 status code to acknowledge the receipt of the event.
03	`createOrder` function	This function is called when a `checkout.session.completed` event is received. It retrieves the necessary information from the event and customer details. It generates a unique order ID, creates an order object with relevant details (such as intent ID, amount, customer, items, etc.), and stores it in the `orders` collection in the database. It also calls the `deleteCart` function to delete the cart items associated with the user. Finally, it responds with a 200 status code to indicate the successful creation of the order.
04	`deleteCart` function	This function is called by the `createOrder` function to delete the cart items associated with the user. It receives the user ID and an array of cart items as parameters. It iterates over the cart items and deletes each item from the `cartItems` collection in the database. This function is responsible for clearing the user's cart after the order is created.

2.5 DashBoard Home

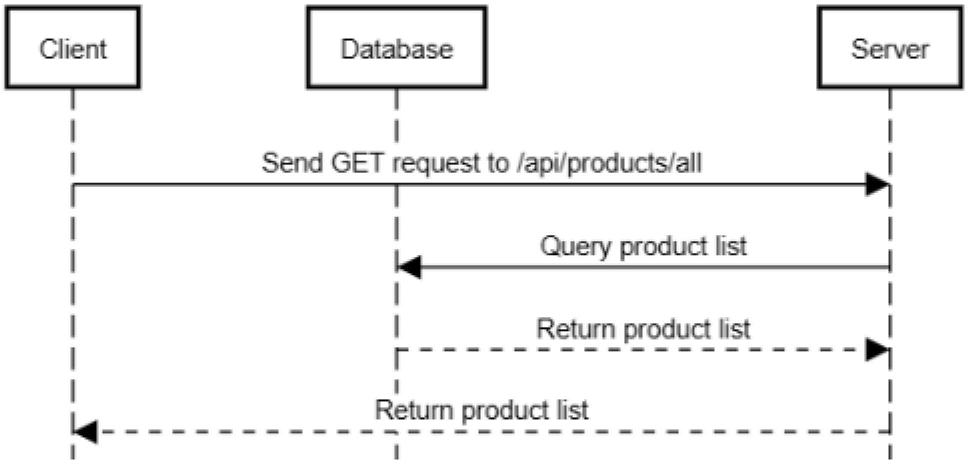
a. Class Diagram



b. Class Specfication

No	Function	Description
1	`getAllOrder`	Fetches all orders from the API.
2	`getAllProducts`	Fetches all products from the API.
3	`setAllProducts`	Dispatches an action to set the fetched products in the Redux store.
4	`setOrders`	Dispatches an action to set the fetched orders in the Redux store.
5	`CChart`	A component from the CoreUI library used to render different types of charts.
6	`useSelector`	A Redux hook used to access data from the Redux store.
7	`useDispatch`	A Redux hook used to dispatch actions to the Redux store.
8	`useEffect`	A React hook used to perform side effects, such as fetching data, when the component mounts or updates.

c. Sequence Diagram(s)

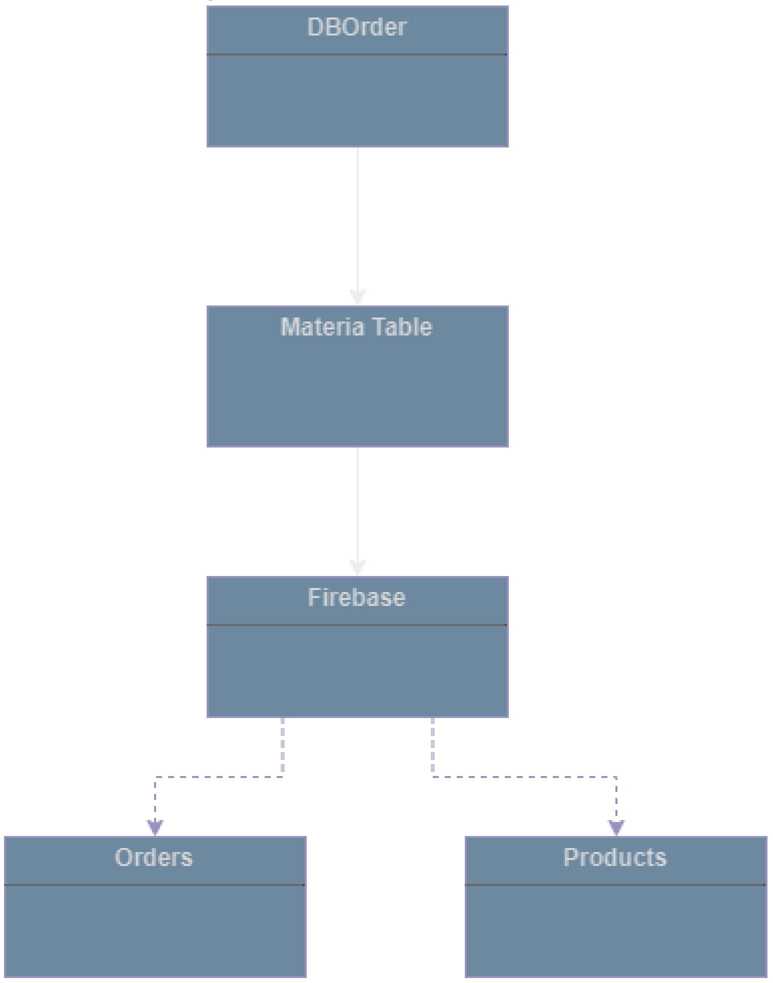


d. API queries

No	Function	Description
1	`getAllOrder`	Fetches all orders from the API using axios. It sends a GET request to the specified API endpoint and returns the data received from the API. If an error occurs during the request, it returns `null`.

2.7 DashBoard Orders

a. Class Diagram

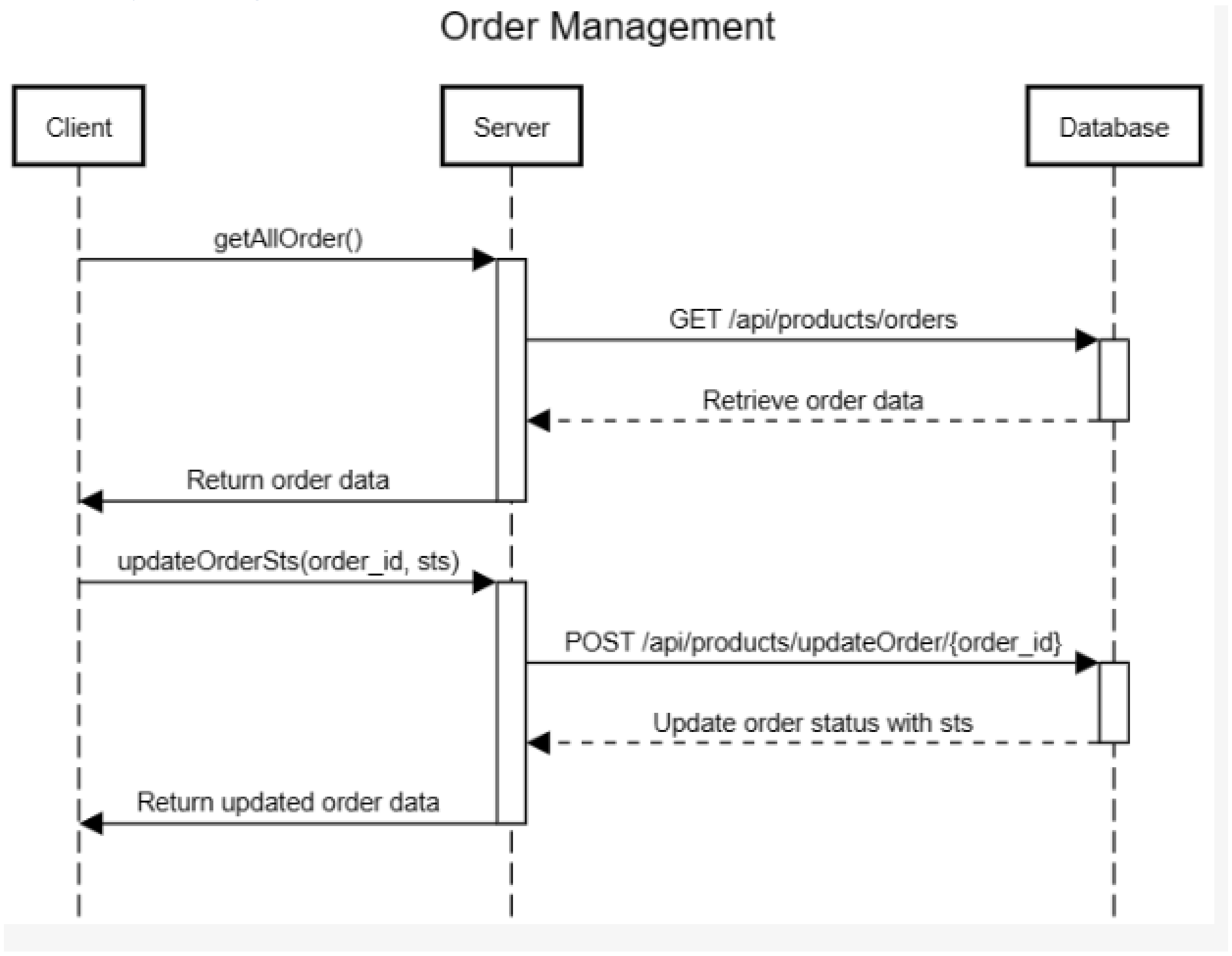


b. Class Specfication

DBOrder.jsx

No	Function	Description
1	DBOrders	Renders a component for displaying orders
2	useEffect	Executes a function after component mount
3	useDispatch	Returns a reference to the store's dispatch function
4	useSelector	Selects a value from the Redux store
5	getAllOrder	Calls an API to fetch all orders
6	setOrders	Updates the orders state in Redux store
7	OrderData	Renders a component for order data display

c. Sequence Diagram(s)

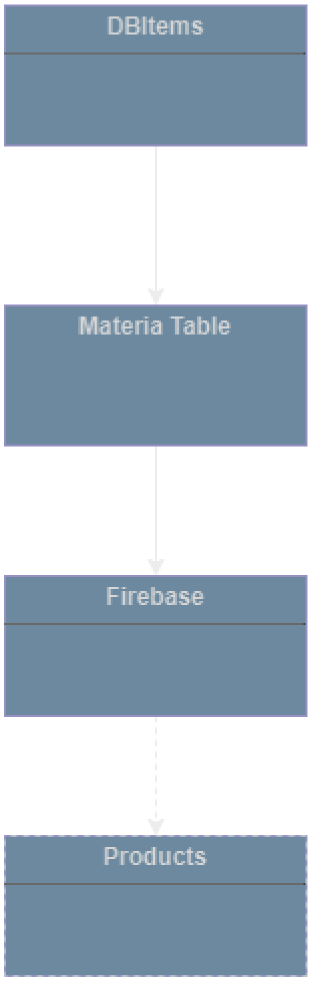

Order Management
Client
Server
Database
getAllOrder()
GET /api/products/orders
Retrieve order data
Return order data
updateOrderSts(order_id, sts)
POST /api/products/updateOrder/{order_id}
Update order status with sts
Return updated order data

d. API queries

No	Function	Description
1	getAllOrder	Retrieves all orders from the API.
2	updateOrderSts	Updates the status of an order in the API.
3	renderTable	Renders a table component for displaying data.
4	formatDate	Formats a date string into a desired format.
5	calculateTotal	Calculates the total amount based on order items.
6	validateInput	Validates user input for order form.
7	handleDelete	Deletes an order from the API.
8	handleEdit	Modifies an order in the API.
9	handleAdd	Adds a new order to the API.

2.8 DashBoard Items

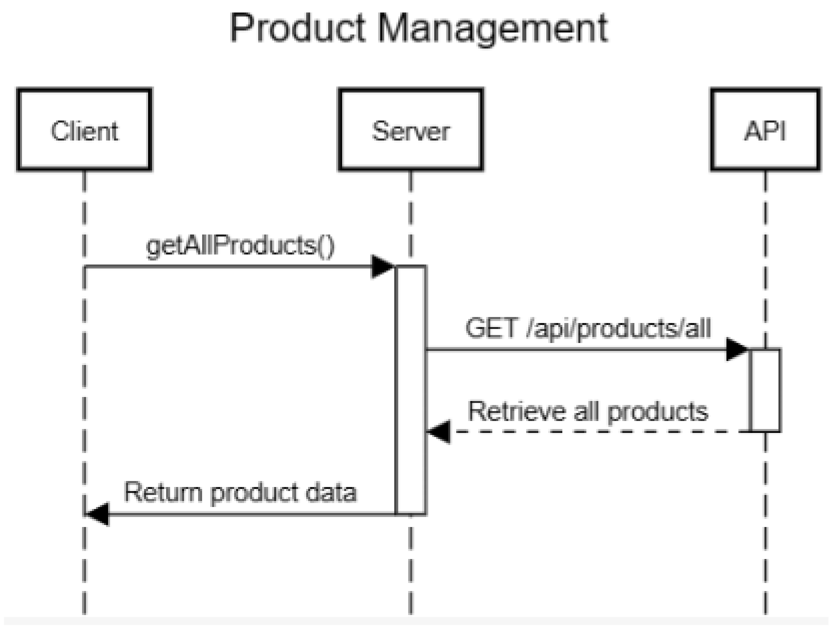
a. Class Diagram



b. Class Specfication

No	Function	Description
1	DBItems	Renders a component for managing and displaying a list of products in a table format.
2	useSelector	Retrieves the **`products`** state from the Redux store.
3	useDispatch	Accesses the dispatch function from the Redux store.
4	deleteAProduct	Calls the API to delete a product based on the provided **`productId`**.
5	getAllProducts	Fetches all products from the API.
6	MdAttachMoney	Icon component for displaying a money symbol.
7	DataTable	Custom component responsible for rendering a table with various functionalities.
8	alertNULL	Dispatches an action to clear/nullify an alert message.
9	alertSuccess	Dispatches an action to display a success alert message.
10	setAllProducts	Updates the **`products`** state with new data in the Redux store.

c. Sequence Diagram(s)

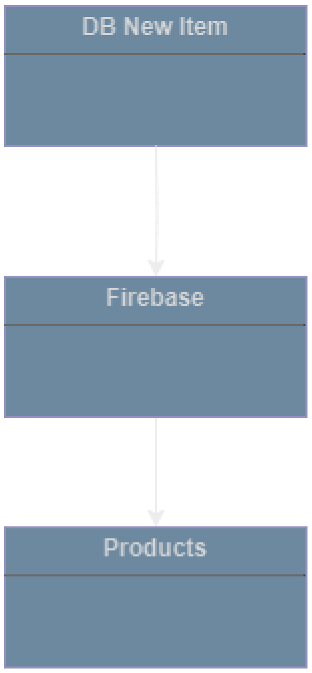


d. API queries

No	Function	Description
1	getAllProducts	Retrieves all products from the API.

2.9 DashBoard New Items

a. Class Diagram

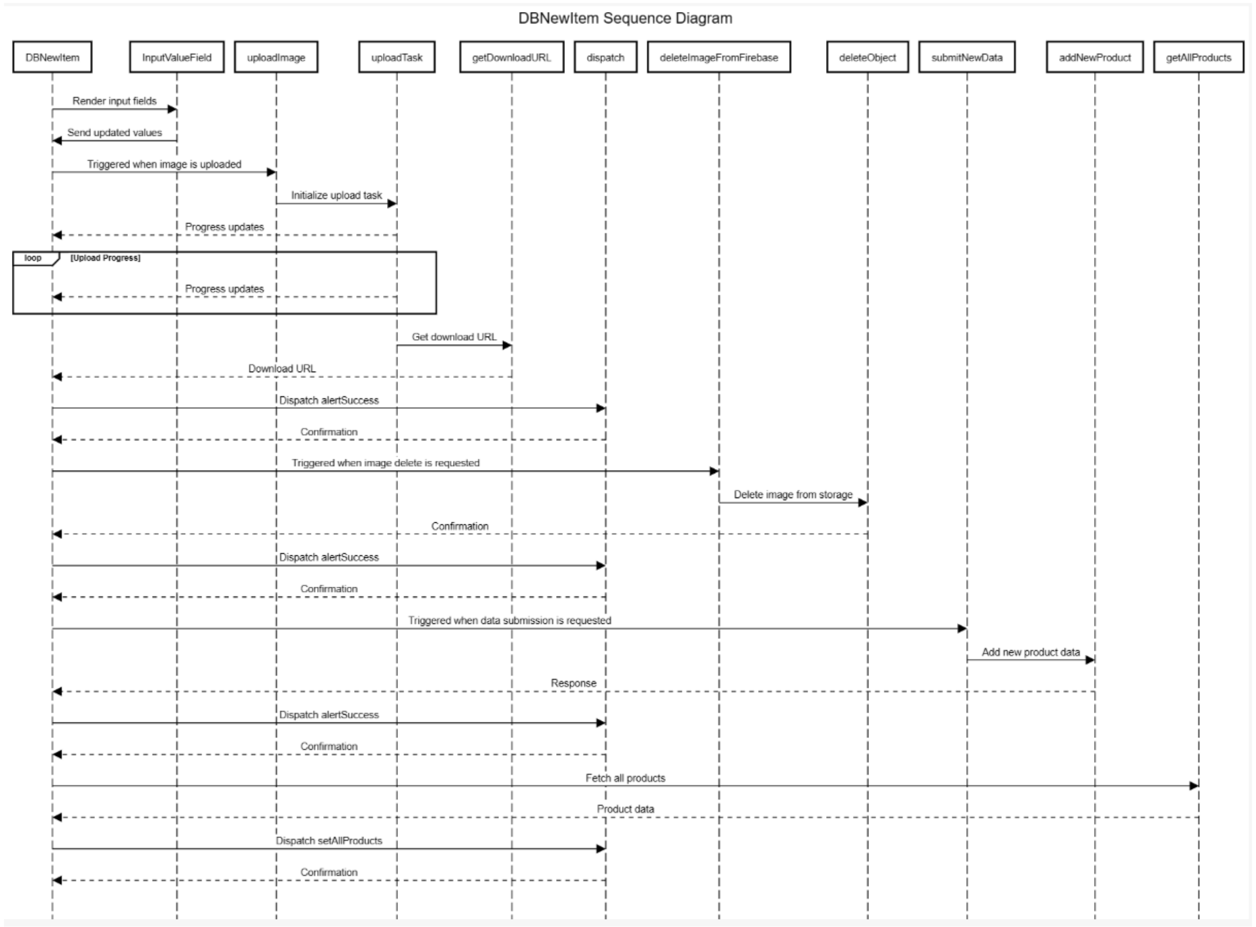


b. Class Specfication

No	Function	Description

1	DBNewItem	The main component for creating a new item in the database.
2	uploadImage	Handles the upload of an image file to Firebase storage.
3	deleteImageFromFirebase	Deletes an image file from Firebase storage.
4	submitNewData	Submits the new item data to the API and updates the Redux store.
5	InputValueField	A reusable input field component.

c. Sequence Diagram(s)

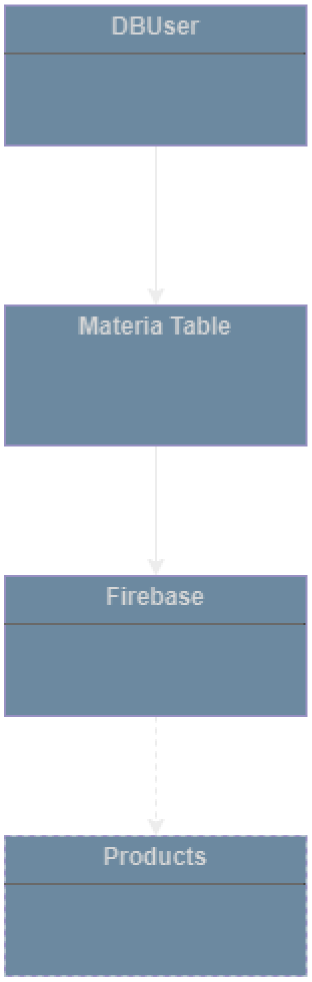


d. API queries

No	Function	Description
1	addNewItemToCart	Add new items to the cart

2.10 DashBoard User

a. Class Diagram

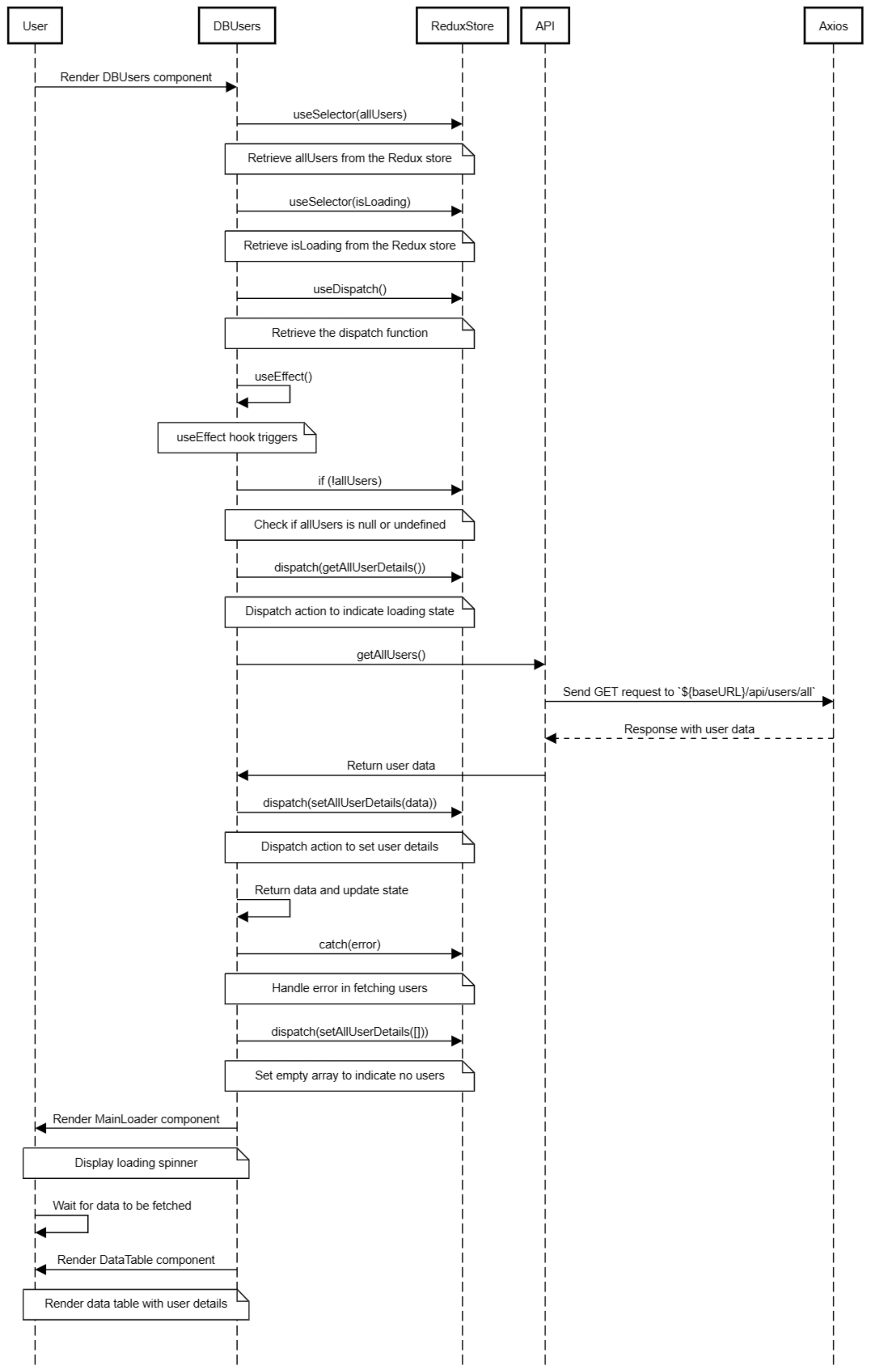


b. Class Specfication

No	Function	Description
1	\`useSelector\`	A React Redux hook that selectively extracts data from the Redux store.
2	\`useDispatch\`	A React Redux hook that returns the \`dispatch\` function.
3	\`useEffect\`	A React hook that allows executing side effects in functional components.
4	\`getAllUsers\`	A function imported from the \`api\` module that fetches all users.
5	\`getAllUserDetails\`	An action creator function from the \`allUsersAction\` module.
6	\`setAllUserDetails\`	An action creator function from the \`allUsersAction\` module.
7	\`MainLoader\`	A component that displays a loading spinner.
8	\`useState\`	A React hook that adds state to functional components.

9	`dispatch`	A function used to dispatch actions to the Redux store.
10	`dispatch(getAllUserDetails())`	A dispatch action that indicates the loading state.
11	`getAllUsers().then()`	Fetches all users and returns the promise with the data.
12	`.catch()`	Handles error in fetching users and sets an empty array in case of an error.
13	`<MainLoader />`	Renders the `MainLoader` component if `isLoading` is true.
14	`<img />`	Renders an image tag with the user's photo or a default avatar.
15	`<DataTable />`	A custom component that displays tabular data.

c. Sequence Diagram(s)

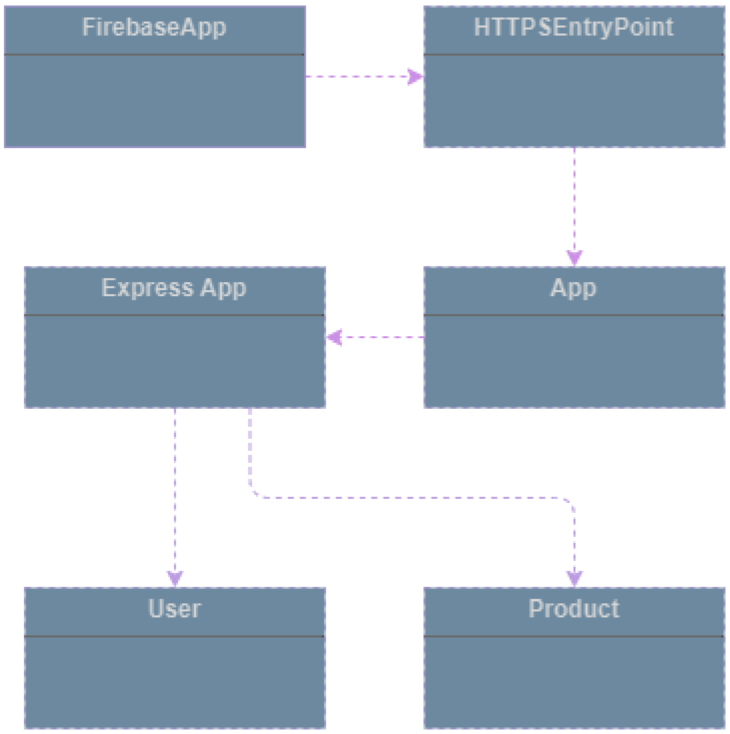

User
DBUsers
ReduxStore
API
Axios
Render DBUsers component
useSelector(allUsers)
Retrieve allUsers from the Redux store
useSelector(isLoading)
Retrieve isLoading from the Redux store
useDispatch()
Retrieve the dispatch function
useEffect()
useEffect hook triggers
if (!allUsers)
Check if allUsers is null or undefined
dispatch(getAllUserDetails())
Dispatch action to indicate loading state
getAllUsers()
Send GET request to `${baseURL}/api/users/all`
Response with user data
Return user data
dispatch(setAllUserDetails(data))
Dispatch action to set user details
Return data and update state
catch(error)
Handle error in fetching users
dispatch(setAllUserDetails([]))
Set empty array to indicate no users
Render MainLoader component
Display loading spinner
Wait for data to be fetched
Render DataTable component
Render data table with user details

d. *API queries*

No	Function	Description
1	`` `getAllUsers` ``	An asynchronous function for retrieving all users.
2	`` `axios.get` ``	A function from the Axios library that sends a GET request to the specified URL.
3	`` `${baseURL}/api/users/all` ``	The endpoint for retrieving all users.
4	`` `try` ``	A block of code to be executed that may throw an error.
5	`` `const res` ``	A variable that stores the response from the API call.
6	`` `res.data.data` ``	The user data extracted from the response object.
7	`` `catch` ``	A block of code to be executed if an error is thrown in the try block.
8	`` `err` ``	An error object that is caught in case of an error in the try block.
9	`` `return null` ``	A statement that returns null when an error occurs.

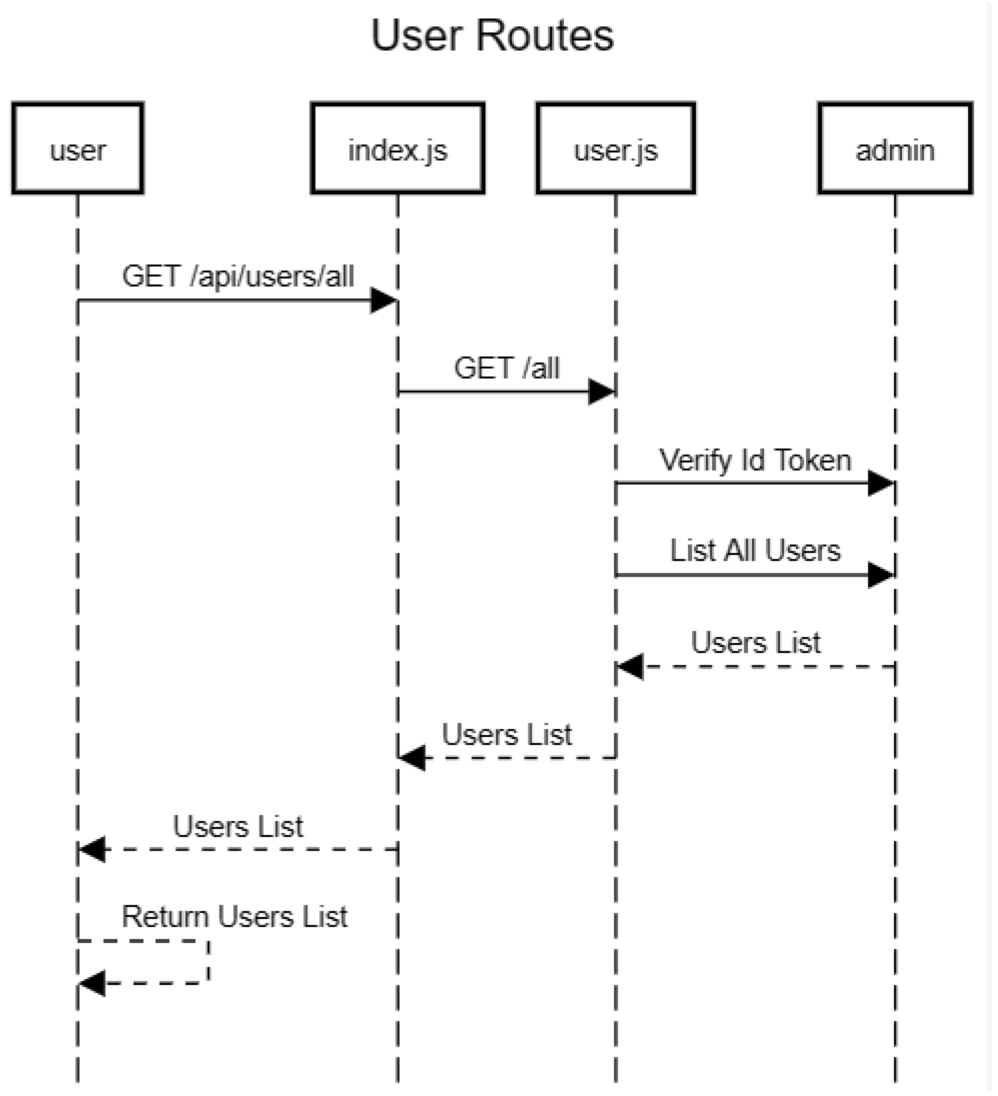
3. Server

a. Class Diagram

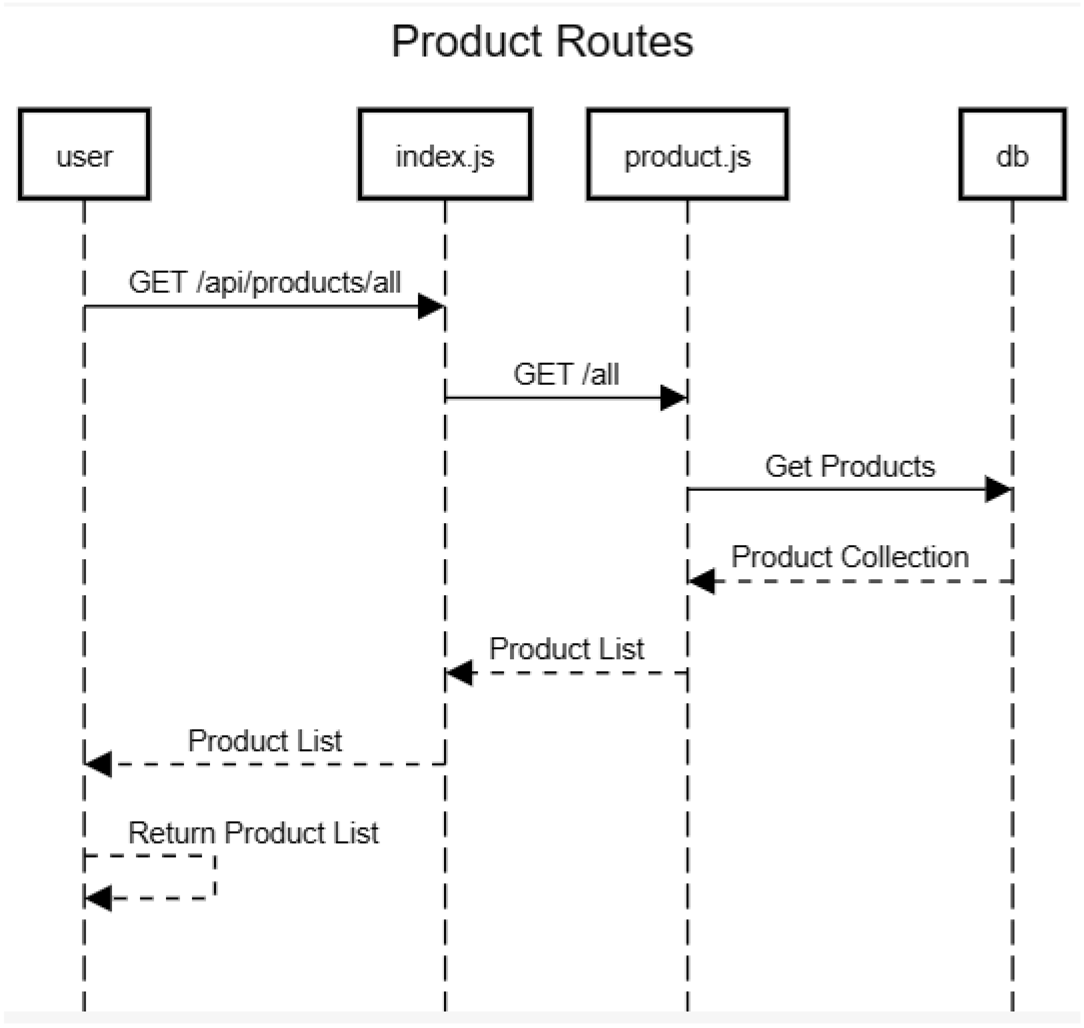


b. Sequence Diagrams

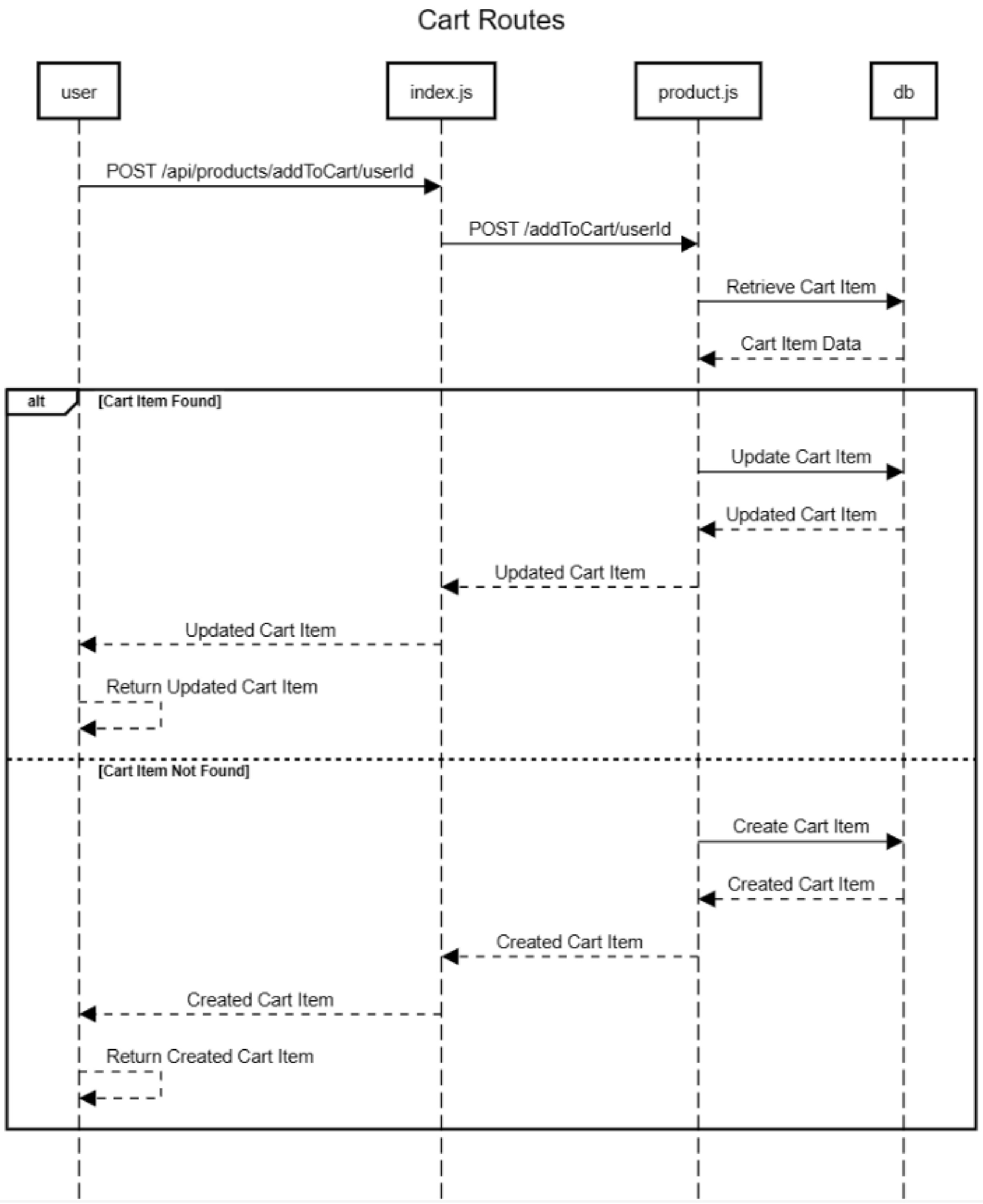
User Routes


User Routes
user
index.js
user.js
admin
GET /api/users/all
GET /all
Verify Id Token
List All Users
Users List
Users List
Users List
Return Users List

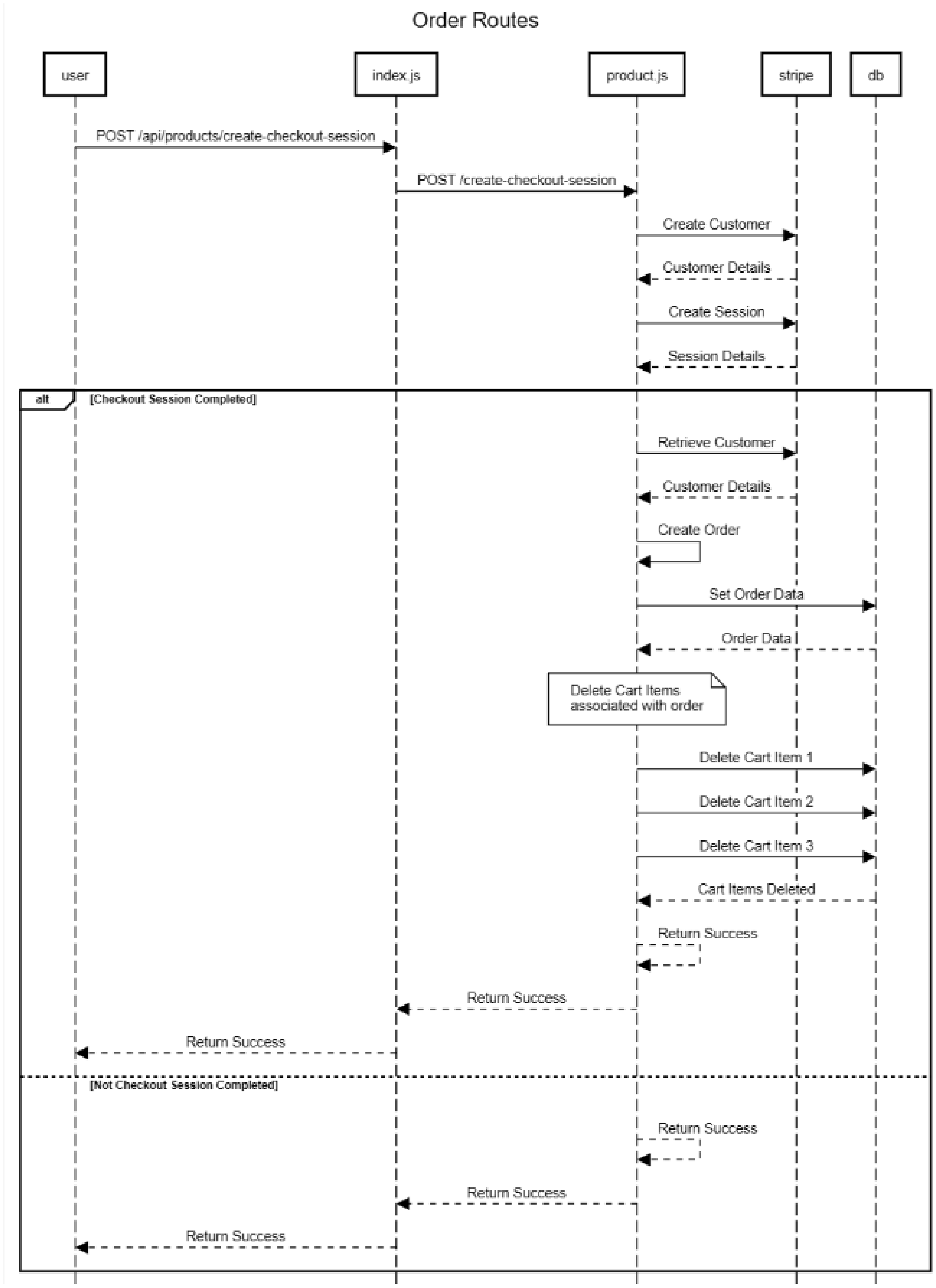
Product Routes



Cart Routes



OrderRoutes


Order Routes
user
index.js
product.js
stripe
db
POST /api/products/create-checkout-session
POST /create-checkout-session
Create Customer
Customer Details
Create Session
Session Details
alt
[Checkout Session Completed]
Retrieve Customer
Customer Details
Create Order
Set Order Data
Order Data
Delete Cart Items associated with order
Delete Cart Item 1
Delete Cart Item 2
Delete Cart Item 3
Cart Items Deleted
Return Success
Return Success
Return Success
[Not Checkout Session Completed]
Return Success
Return Success
Return Success

c. Class Specification

Index.js

No	Function	Description
1	FirebaseFunctionsApp	Represents the Firebase Functions application.
2	ExpressApp	Represents the Express.js application.
3	ExpressRouter	Represents an Express.js router.
4	UserRouter	Represents the router for user-related routes.
5	ProductRouter	Represents the router for product-related routes.

Product.js

No	Route	Description
1	POST /create	Create a product in the Firestore database.
2	GET /all	Get all products from the Firestore database.
3	DELETE /delete/:productId	Delete a product from the Firestore database based on the product ID.
4	POST /addToCart/:userId	Add a product to the user's cart in the Firestore database.
5	POST /updateCart/:user_id?productId=&type=	Update the quantity of a product in the user's cart in the Firestore database.
6	GET /getCartItems/:user_id	Get all items in the user's cart from the Firestore database.
7	POST /create-checkout-session	Create a checkout session for Stripe payment.
8	POST /webhook	Handle Stripe webhook events.
9	GET /orders	Get all orders from the Firestore database.
10	POST /updateOrder/:order_id?sts=	Update the status of an order in the Firestore database.

User.js

No	Route	Description
1	GET /	Return a response indicating that the router is inside the user router.
2	GET /jwtVerification	Verify a JSON Web Token (JWT) provided in the Authorization header. Returns the decoded value of the token if it is valid.
3	GET /all	List all users from the Firebase Authentication service. Returns an array of user data.

IV. Database Tables

1. Authentication

Field name	Type	Size	Unique	Not Null	PK/FK	Note
Identifier	string	255		X		
Providers	string	255		X		
Created	date			X		Creation date
Signed In	date			X		Sign-in date
User UID	string	50		X		

2. Products

Field name	**Type**	**Size**	**Unique**	**Not Null**	**PK/FK**	**Notes**
imageURL	string	MAX		X		URL of the image
productId	string	50	X	X	PK	Unique identifier for the product
product_category	string	255		X		Category of the product
product_name	string	255		X		Name of the product
product_price	string	50		X		Price of the product
quantity	integer	100		X		Quantity of the product

3. Cart Items

! Get data with product

Field name	**Type**	**Size**	**Unique**	**Not Null**	**PK/FK**	**Notes**
imageURL	string	MAX		X		URL of the image
productId	string	50	X	X	PK	Unique identifier for the product
product_category	string	255		X		Category of the product
product_name	string	255		X		Name of the product
product_price	string	50		X		Price of the product
quantity	integer	100		X		Quantity of the product

4. Orders

Field name	Type	Size	Unique	Not Null	PK/FK	Notes
amount	String	50		X		Amount of the transaction
created	String	50		X		Creation timestamp of the transaction
customer	String	50		X		Customer information
address	String	255		X		Address of the customer
city	String	50		X		City of the customer
country	String	50		X		Country of the customer
line1	String	255		X		Address line 1
line2	String	255				Address line 2
postal_code	String	10		X		Postal code of the address
state	String	6		X		State of the address
email	String	50		X		Email of the customer
name	String	50		X		Name of the customer
phone	String	10		X		Phone number of the customer
tax_exempt	String	10				Tax exemption details
tax_ids	String	10				Tax identification numbers
intentId	String	10	X	X	PK	Intent ID
items	String	10		X		Items in the transaction
imageURL	String	MAX		X		URL of the image associated with the item
productId	String	50		X	FK	Unique identifier for the product
product_category	String	255		X		Category of the product
product_name	String	255		X		Name of the product
product_price	String	50		X		Price of the product
quantity	integer	100		X		Quantity of the product
orderId	String	10		X		Order ID
payment_method_types	String	10		X		Payment method types
shipping_details	String	255				Shipping details
status	String	50		X		Status of the transaction
sts	String	50		X		Status of the delivery
total	String	50		X		Total amount of the transaction
userId	String	50		X	FK	User ID