1. **Button.** This is the first component we can see when going through the list of components. This component, as the name would imply, is a button. This allows us to create and make buttons throughout our project with the same functionality, appearance and footprint. By footprint, I mean how many lines of code it will take up. It allows our code to remain tidier and makes it less of a hassle to read through and understand.

* 1. Several places where button was used:
     1. Admin Product Creation Page
     2. Sign-in/Authentication Pages
     3. Order Cart
     4. Menu/Product Page

2. **CartScreen**:

* Purpose: Represents the screen displaying the contents of the user's shopping cart.
* Involvement :
  + Utilizes the usechart hook from cartprovider to fetch cart items, total amount, and handle checkout functionality.
  + Renders a Flatlist of cart items using CartListItems component for each item.
  + Displays total amount and a checkout button.
  + Manages the status bar appearance based on the platform using stausbar from Expo.
* Necessity: Essential for providing a user interface to view and interact with the shopping cart.

3. **Link Component**:

* Purpose: Represents a link or navigation element within the application.
* Interactions:
  + Navigates users to different screens or URLs based on the specified href prop.
  + Styled to resemble a button for better user interaction.
* Necessity: Allows users to navigate between different screens or sections of the application easily.

4. **Remote image Ccomponent:**

* Purpose: The remote image component is designed to render images from remote sources, particularly from the Supabase storage.
* Interactions:
  + Receives props including path and fallback (a fallback image URL).
  + Utilizes the Supabase client to download the image from the specified path.
  + Converts the downloaded image data to a base64 string for rendering in the image component.
* Necessity: Necessary for rendering images stored remotely in the application, providing flexibility and scalability in image management.

#### **Image component:**

* Purpose: The image component is a UI component provided by React Native for rendering images.
* Interactions:
  + Receives props such as source to specify the image source Renders the image in the UI based on the provided source.
* Necessity: Essential for displaying images within the application's user interface.

5. **Productlistitem.** This is a great useful component that handles displaying product images. When a user is scrolling through a menu, the small, pressable images that they see are due to this component. This handles the default and set image for a product, as well as the pricing and name for the product.

1. Examples of ProductListItem in use:
   1. Menus: Displays all products
   2. Orders page: Displays products

6. **Orderlistitem.** As the name would imply, this is used to list orders. Both Admin and User side. Not to be confused with OrderItemListItem, this one only displays the orders in a list format for FlatList. It is also used to display the order id/name when opening the order’s details page.

1. Examples of OrderListItem in use:
   1. Order details page: Admin and User side
   2. Admin Orders List
   3. Archived Orders List: List of previous orders

7. **EditScreenInfo:**

* **Purpose:** Used to demonstrate how to structure a screen or a section of a screen by providing placeholder content and basic styling. It typically includes elements like text, buttons, or input fields.
* **Interactions:** interacts with text, themes,buttons, on the home screen and more
* **Necessity:** can greatly benefit development workflow, code organization, and collaboration, especially in larger or team-based projects.

8. **StyledText:**

* **Purpose:** allows developers to apply styles such as font size, color, font weight, alignment, etc., to text elements within their application. This is useful for maintaining consistent styling throughout the app, improving readability, and adhering to design guidelines.
* **Interactions:** works with buttons, themes, and all pages
* **Necessity:** they offer significant benefits in terms of consistency, maintainability, reusability, productivity, adaptability, clarity, and integration.

9. **Themed**

* **Purpose:** using a theme to style components consistently across an application. It typically involves defining a set of styles or colors that can be applied to various UI elements throughout the app.
* **Interactions:** interacts with all pages
* **Necessity:** are essential for building maintainable, scalable, and visually consistent React Native applications, making them a necessity for modern app development workflows.

**Architecture Overview**

**Admin Front-End** - The Admin Side is inaccessible to a normal user, and must be managed through the database/backend of the application. There a user can be assigned the role of Admin allowing them to access both the User and Admin side of the application. On the Admin side, the front end affords the Admin the ability to create and manage the menu. This means they can create, delete and update the menu. They are also able to access current and previous order information, with the former allowing the Admin to manage things like the status of the order. In example, the Admin can update a user that their order is being prepared, ready for pickup or out for delivery.

When it comes to state in our application, this is managed by our database as well as through Stack Navigator. Our backend allows us to control what a user sees and has access to. The biggest example being administrative roles. If a user does not have the correct role, they cannot access, let alone see the Admin section of the app. Our backend is also used to save a user’s order information and login information. Without that, a user would not be automatically logged into the app should they close it or move to another app. It would also force them to restart their order if they move a way for a moment to, for example, answer a call. Stack Navigator is also used to help manage how the user navigates through the app. With that, we can set how a user can reach a certain screen, as well as how they backtrack through their accessed screens. React Context is used within the app to help manage state as well. It works in tandem with the backend to authenticate a user. It is also used to help manage the state of a user’s cart/order. Since the simplest purpose of React Context is to share data within the application globally, we are able to use it to manage an order from the context of an Admin and user.

Some of the architectural choices made throughout our project, such as components, were made to make management and creation simple. Using components allows us to implement new pages throughout our app with ease. It also allows us to access and manage data easily. Context making data global lets us manage data from different areas within our application without having to spend time rewriting code to suit the next context.

Additionally, React Context emerges as a vital component for state management, playing a pivotal role in user authentication and order management workflows. This powerful tool operates synergistically with the backend infrastructure, enabling streamlined authentication processes and facilitating centralized order management. Leveraging React Context's inherent capability to share data globally within the application, we empower Admins and users alike to seamlessly manage order-related tasks from their respective contexts. Architectural decisions, including the widespread adoption of components, further enhance the project's manageability and scalability. By leveraging components, the development process is streamlined, allowing for the effortless integration of new pages and the efficient management of data. The utilization of Context further amplifies development efficiency, enabling data to be seamlessly shared and managed across diverse application contexts without necessitating extensive code rewriting.