# RESUME

* Used interfaces for props.
* Established a set of distinct features for server and client components.
* Made a lot of reusable components.
* Used React.js with TypeScript with Tailwind CSS to code the frontend of the application.
* We created .env.local file to store the API keys from the supabase.
* Made environment variables that store next\_public\_supabase\_url, and more.

# Installs

* npm install react-icons
* npm install tailwind-merge
* npm install [supabase@”>=1.8.1](mailto:supabase@”%3e=1.8.1)” –save-dev
* npx supabase login

# sidebar component

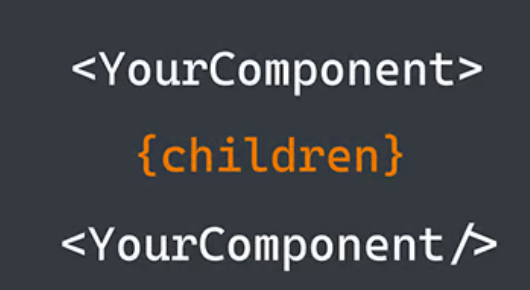
* Created a folder called “**components**”.
* Create a file called “Sidebar.tsx” and made stateless function inside. *(Code snippet: sfc)*

## ReactNode

* **React.ReactNode** represents an array of react elements, Boolean, number, or a string.
  + It defines the type of data a variable can hold inside.
* Most of the time a JSX component is closed.

< Component />

* + But we use the {children} prop when the component is not closed.



## Interace Sidebarprops

* “**interface** **SidebarProps**” basically is type checking mechanism. In TypeScript, interfaces are used to describe the structure of objects or classes. They define a set of properties and their types that an object must have in order to be considered compatible with the interface.

## React.fc

* **React.FC** is used to define the functional component. It is not mandatory and there are other ways to define a functional component.
  + Basically, it provides additional type checking.
  + So, the props that are coming to the component must be of type {children} which is of type ReactNode.
  + It’s not universally used as it provides additional constraints and non-standard props cannot be passed.

## Server Components

* **Server Components** get rendered on the server itself and are static.
  + They don’t need interaction or don’t need any state and don’t change are server components.
  + They cannot use useEffect, useStates etc. inside of a server component.
  + All the API calls, database calls happen on the server side, and it makes the site faster.
  + Environment is kept a secret on the server.
* **“use client”** is used as a convention. It declares a boundary between a server and a client component. Basically, we use “use client” where we need to use react hooks. The files using “use-client” are considered to be client-side components.
  + It is the proper way to pass server components inside of a client component.

## use memo

* This hook is used to memoize the result of a computation and use it for future purpose.
* Since the dependency array is empty, “[ ]”, the routes are only computed once and are reused in the subsequent renders optimizing performance.
* Basically, the routes are not rendered again and again because they don’t change throughout the application.

# Header Component

# Button component

## forward ref

* ‘forwardRef’ function is used to forward a ref to a child component, allowing parent components to interact with DOM elements or components defined within the child component.
* The button component uses a forwardRef which allows the ref to be forwarded from the parent component to the underlying <button> element.

# supabase

* We created .env.local file to store the API keys from the supabase.
* Made environment variables that store next\_public\_supabase\_url, and more.
* Supabase is excellent because it provides us a variety of features like table editor, sql editor, database viewer and has a separate tab for authentication.
* We can use supabase for resetting the password of the user, removing the user, email templates, providers etc.
* Stripe Subscription Template provided by Supabase was used to maintain a database for the user, our songs etc.
* This template is used to create table that are handy for managing Stripe payments and user database.
* There are row level securities that ensure that only the user can view, update, delete their data and no one else. This is done with the help of auth.id().
* Created tables using the table editor of the Supabase with all the required entries determining the primary and foreign keys.
* Create RLS policies that provide policies for read access, insert access, delete access etc.
* Created public storage buckets to upload and store our songs.
* Made personal access tokens with Supabase CLI to ensure permissions.
* Generated types for typescript using Supabase CLI so that we can generate a file that includes all the database in typescript language.