# Filtering Cabins

* The filter / sort will set the URL STATE
* This will be available from everywhere, so we don’t have to make the Cabins Table a Children of the Filter/Sort Component

# Sort Cabins – sort component

* A dropdown component to allow the user to select by what to sort the Cabins Table
* I will use the MENUS compound component to build this !!!

# Bookings Table

* We get the Bookings table data from Supabase into the app

## Server-Side Filtering of the Bookings Table

* We want to receive Straight from the API only the FILTERED bookings, not all of them and then filter on the client side (like the cabins)
* We will adjust the getBookings() API call to get only the filtered data
* We use the URL state to know how to filter the data by status
* We **PASS the status into the getBookings API call** and filter depending on what the status is

## Server-Side Sorting of the Bookings Table

* We are using the same approach as Server Side Filtering
* We adapt the query for the getBookings with the .order() function

# Pagination

* A simple component to set the current page state to the URL and fetch the a number of rows
* Will approach the same way as the filter and sorting
* We use the **{count:’exact’} query** to get the number of **ALL rows**
* We use **.range(start, end) to** get the exact rows we need

# Prefetching with React Query

* We Fetch the next page before it is actually displayed

### What we do:

* We need the queryClient ( **useQueryClient()**)
* We call **prefetchQuery()** on the **queryClient**
  + **queryClient.prefetchQuery()**
* A screen shot of a computer program

  Description automatically generated

# Booking Detail Page

* We add a **Menu** for each Boooking, and one of the options will take us to the **Booking detail page**
* We have already placed the **entire booking table inside the** Menus component
* We add the Toggle and the Button for the **Booking Details**
* We implement a new **ROUTE** and a new **PAGE**

# Check In a Booking

## Requirements

* **Check In/ Check Out**
  + Users should be able to
    - Delete booking
    - Check in booking
    - Checkout out booking
  + Bookings may not have been paid at guest arrival. Therefore, on check in, users need to accept payment (outside the app) and then confirm the payment has been received (inside the app)
  + On check in, the guest should have the ability to add breakfast for the entire stay, if they hadn’t already

# Allow the guest to add Breakfast while checking in

* We will add a new checkbox for the breakfast, in the check in flow

# Authentication

* We will use Supabase to **AUTHENTICATE users**
* WE must create the 1st user in Supabase
  + In the Authentication tab
  + We can use lots of 3rd party providers for login
* We create a new Service – apiAuth.js **(for authentication)**
  + We are using the Login With email/password API
  + We pass an **object with email and pass**
    - Common in MODERN app development
  + After success login, supabase stores the user data in **session storage**
* We create a new custom hook – **useLogin** for all the React Query login logic
  + We are using a **MUTATION (something changes in the server, the user gets authenticated)**
  + It’s easier to handle the success and the error states
  + On the **onSuccess,** we receive the data from the apiLogin loginFn, and we can use it
  + On the **onError,** we receive the **error** from the apiLogin (Error) fn

# Protected Routes – Authorization

* We will **WRAP all ROUTES into a PROTECTED ROUTE**
  + We actually wrap only the **AppLayout**
  + We want the Login and the \* (page not found) to be available to everyone

## How to use the Protected Route

1. We need to create the **Protected Route** component
2. A screen shot of a computer program

   Description automatically generated
   1. Like this, all routes in AppLayout are also the childrens of **Protected Route**
3. **Load the authenticated user**
   1. We load the user from supabase with a new **function in the apiLogin**
   2. This means that the user data will be refetched from the API
      1. Check if there is an active session (using the local storage metadata)
      2. 
   3. If there is an active session, we **refetch the data from supabase** – MORE SECURE
   4. A computer screen shot of a program code

      Description automatically generated
   5. .
   6. We will manage this using React Query and useQuery, so a new custom hook – **useUser.js**
      1. We use **useQuery** to store the user data into the cache
      2. And we also create here the isAuthenticated flag
      3. A screen shot of a computer program

         Description automatically generated
4. **Show a SpinnerA screen shot of a computer program

   Description automatically generated**
5. **If no user, redirect to Login page**
   1. A screenshot of a computer

      Description automatically generated
   2. If the user is **Not AUTH**  and the **LOADING is done,** navigate to the **/login** page
6. **If there is a user, render the app**

# User Sign Out

* A small button into the header and like that the user will log out
* We create a new **apiAuth** function for the logout API call
* We create a new custom hook **useLogout()**
  + We are using the mutation
  + Also **REMOVE** the queries
  + Navigate to the ‘/login’ page

# Sign Up Form

* In this app, new users can only be created from inside the APP
* Like this, only **hotel employees** will have access
* We are using the **Users page**
* For **Improved error handling - react Hook Form** library

# RLS – Authorization policies

* We need to update all ROW LEVEL SECURITY policies
* Update ALL policies to **APPLY TO AUTHENTICATED USERS ONLY**

# App Header

* This will contain the
  + Username of the user
  + An avatar picture
  + A small Menu
    - Button for the user toupdate the Profile
    - Logout button

# Update User Details

* We have 2 forms
  + One for user Name and Avatar
  + One for Password
* Both will use the same **custom hook useUpdateUser()**
* This hook is using the **updateCurrentUser() api call**
* Where depending, we either update the Full Name or the Password. But not both at the same time
* We also update the avatar, if the is a new image uploaded
  + Create a unique name for the avatar
  + Upload to the storage
  + Re UPDATE the user with the new avatar file name

# Dark Mode using CSS Variables

* We will use the **DARK MODE css variables**  from the GlobalStyles.css file
* We are using the **DARK MODE**  variables when the class of the **ROOT** element will be dark
* In the **GlobalStyles.js**
  + We have added the &.light-mode {}
  + And &.dark-mode{}

A screen shot of a computer

Description automatically generated

* When there is no class at all 🡺 **&**
* When the class Is light-mode 🡺 **&.light-mode**

## Managing the state

* We are going to use the Context API to manage the **GLOBAL STATE of the theme**
* We create a **Context Folder,** where we store all the contexts
  + DarkModeContext
  + DarkModeProvider
  + Custom hook 🡪 useDarkMode

## To set the className on the :root element

* We are using an **useEffect** in the **context** to set the className of the root element