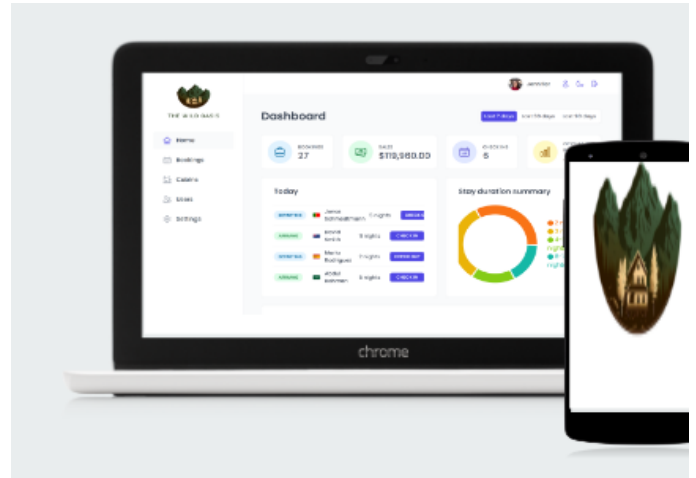




# THE WILD OASIS

# The Wild Oasis



## Overview

Develop a comprehensive internal application for Wild Oasis, a boutique hotel with 8 cabins, to manage **bookings, cabins, and guest interactions** efficiently. This system will also be prepared to integrate with a future customer-facing website that will use the same backend API.

## Goals

They need a custom-built application to manage everything about the hotel: **bookings, cabins, and guests.**

1. **Employee Authentication:** Ensure that only hotel employees can access and operate the application.
2. **Cabin Management:** Enable employees to manage cabin details including creation, updates, and deletion.
3. **Booking Management:** Allow employees to manage all aspects of bookings from creation to check-out.
4. **Payment Confirmation:** Handle on-site payment confirmations within the app.
5. **Operational Dashboard:** Provide a dashboard to quickly access important data and actions related to daily hotel operations.
6. **Customization and Settings:** Allow configuration of hotel-specific settings such as pricing and booking constraints.
7. **User Experience:** Implement a user-friendly interface with dark mode support.

## Specifications

- **User Authentication:**

- Secure login/logout functionality for hotel staff.
- User profile management including avatar uploads and personal information updates.

- **Cabin Management:**

- A detailed view of all cabins with features like photo, name, capacity, price, and discounts.
- Options to add, edit, and delete cabin entries.

- **Booking Management:**

- A comprehensive booking table displaying details such as arrival/departure dates, status, and payment information.
- Functionality to check-in and check-out guests, along with the ability to update booking statuses and manage guest additions like breakfast.

- **Payment Handling:**

- Option to confirm payment received after check-in directly within the app.

- **Operational Dashboard:**

- Displays crucial daily operations such as guest check-ins/outs.
- Visuals and statistics on bookings, sales, occupancy rates, and stay durations.

- **Configurability:**

- Settings for defining application-wide parameters like breakfast price and booking limits.

- **User Interface:**

- Responsive design with dark mode capability.

## Milestones

### Requirement Gathering and Analysis Ph:

Activities: Finalize detailed requirements with stakeholders, and establish system specifications.

### System Design:

Activities: Design the architecture for both the application and the API. Prepare mockups for the user interface.

### Development Phase I -Technology Decisions:

Activities: Decide on what libraries to use.

### Development Phase II - Backend API:

Activities: Develop the API to handle user authentication, cabin and booking management.

### Development Phase II - Frontend Application:

Activities: Implement the frontend interface using the designed mockups and connect it with the same backend API.

\*\*\*\*\* STILL UNDER PROCESSING \*\*\*\*\*

### Testing and Quality Assurance:

Activities: Perform comprehensive testing including unit tests, integration tests, and user acceptance tests. Ensure all functionalities meet the defined criteria.

### Deployment and Training:

Activities: Deploy the system on the required platforms. Train hotel staff on how to use the application effectively.

### Post-Deployment Support and Iteration:

Activities: Provide support for any issues. Plan additional features based on user feedback.

## ❖ Requirement Gathering and Analysis Phase:

### STEP 1:

#### PROJECT REQUIREMENTS FROM THE BUSINESS

- ✦ Users of the app are hotel employees. They need to be logged into the application to perform tasks
- ✦ New users can only be signed up inside the applications (to guarantee that only actual hotel employees can get accounts)
- ✦ Users should be able to upload an avatar, and change their name and password
- ✦ App needs a table view with all cabins, showing the cabin photo, name, capacity, price, and current discount
- ✦ Users should be able to update or delete a cabin, and to create new cabins (including uploading a photo)
- ✦ App needs a table view with all bookings, showing arrival and departure dates, status, and paid amount, as well as cabin and guest data
- ✦ The booking status can be "unconfirmed" (booked but not yet checked in), "checked in", or "checked out". The table should be filterable by this important status
- ✦ Other booking data includes: number of guests, number of nights, guest observations, whether they booked breakfast, breakfast price
- ✦ Users should be able to delete, check in, or check out a booking as the guest arrives (no editing necessary for now)
- ✦ Bookings may not have been paid yet on guest arrival. Therefore, on check in, users need to accept payment (outside the app), and then confirm that 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
- ✦ Guest data should contain: full name, email, national ID, nationality, and a country flag for easy identification
- ✦ The initial app screen should be a dashboard, to display important information for the last 7, 30, or 90 days:
  - ✦ A list of guests checking in and out on the current day. Users should be able to perform these tasks from here
  - ✦ Statistics on recent bookings, sales, check ins, and occupancy rate
  - ✦ A chart showing all daily hotel sales, showing both "total" sales and "extras" sales (only breakfast at the moment)
  - ✦ A chart showing statistics on stay durations, as this is an important metric for the hotel
- ✦ Users should be able to define a few application-wide settings: breakfast price, min and max nights/booking, max guests/booking
- ✦ App needs a dark mode

STEP 1

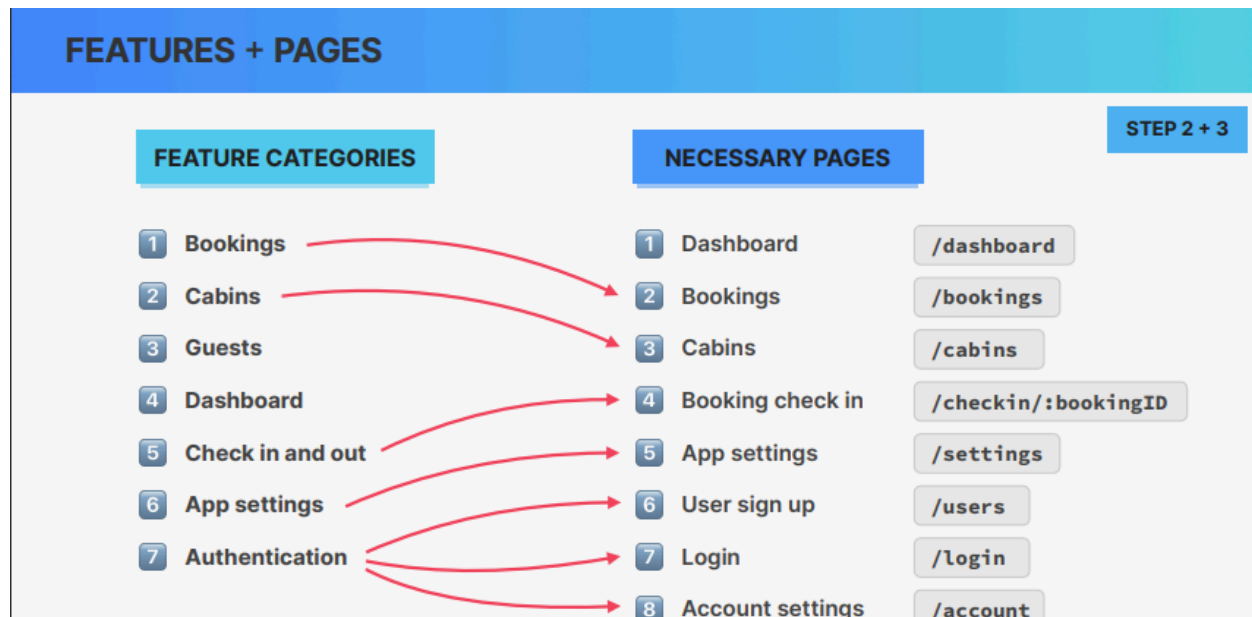
### STEP 2:

#### DIVIDE THE APPLICATION INTO PAGES.

- |   |                |
|---|----------------|
| ✦ Users of the app are hotel employees. They need to be logged into the application to perform tasks  | AUTHENTICATION |
| ✦ New users can only be signed up inside the applications (to guarantee that only actual hotel employees can get accounts)  |                |
| ✦ Users should be able to upload an avatar, and change their name and password  |                |
| ✦ App needs a table view with all cabins, showing the cabin photo, name, capacity, price, and current discount  | CABINS         |
| ✦ Users should be able to update or delete a cabin, and to create new cabins (including uploading a photo)  |                |
| ✦ App needs a table view with all bookings, showing arrival and departure dates, status, and paid amount, as well as cabin and guest data   | BOOKINGS       |
| ✦ The booking status can be "unconfirmed" (booked but not yet checked in), "checked in", or "checked out". The table should be filterable by this important status  |                |
| ✦ Other booking data includes: number of guests, number of nights, guest observations, whether they booked breakfast, breakfast price   | CHECK IN / OUT |
| ✦ Users should be able to delete, check in, or check out a booking as the guest arrives (no editing necessary for now)  |                |
| ✦ Bookings may not have been paid yet on guest arrival. Therefore, on check in, users need to accept payment (outside the app), and then confirm that 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   | GUESTS         |
| ✦ Guest data should contain: full name, email, national ID, nationality, and a country flag for easy identification   |                |
| ✦ The initial app screen should be a dashboard, to display important information for the last 7, 30, or 90 days: <ul style="list-style-type: none"> <li>✦ A list of guests checking in and out on the current day. Users should be able to perform these tasks from here</li> <li>✦ Statistics on recent bookings, sales, check ins, and occupancy rate</li> <li>✦ A chart showing all daily hotel sales, showing both "total" sales and "extras" sales (only breakfast at the moment)</li> <li>✦ A chart showing statistics on stay durations, as this is an important metric for the hotel</li> </ul> | DASHBOARD      |
| ✦ Users should be able to define a few application-wide settings: breakfast price, min and max nights/booking, max guests/booking   |                |
| ✦ App needs a dark mode   | SETTINGS       |

## STEP 2+3:

So, from these seven identified categories we can easily derive the pages that we are going to need (Routes).



## NOTE:

→ Registering users is actually not going to happen on the page called **sign up**, but instead on the page called **users**. The reason for that is that people will not be allowed to simply sign up for the application, instead, it will be inside the app where existing users can register new ones.


## STEP 4:

### TECHNOLOGY DECISIONS.

#### CLIENT-SIDE RENDERING (CSR) OR SERVER-SIDE RENDERING (SSR)?

##### CSR WITH PLAIN REACT

- Used to build **Single-Page Applications (SPAs)**
- All HTML is rendered on the **client**
- All JavaScript needs to be downloaded before apps start running: **bad for performance**
- One perfect use case:** apps that are used "internally" as tools inside companies, that are entirely hidden behind a login



This is exactly what we want to build in this project

##### SSR WITH FRAMEWORK






- Used to build **Multi-Page Applications (MPAs)**
- Some HTML is rendered in the **server**
- More performant**, as less JavaScript needs to be downloaded
- The **React team** is moving more and more in this direction

## NEXT.js

## Remix

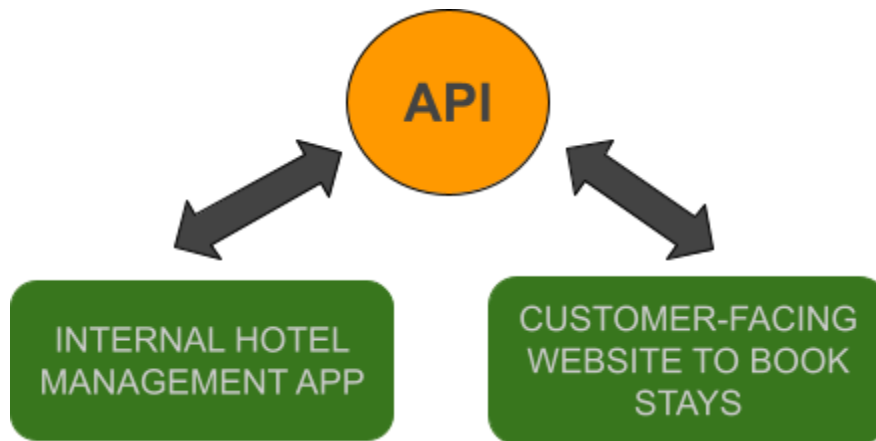
STEP 4

#### TECHNOLOGY DECISIONS

Routing	 <b>React Router</b>	The standard for React SPAs
Styling	<  > styled components	Very popular way of writing component-scoped CSS, right inside JavaScript. A technology worth learning
Remote state management	 <b>React Query</b>	The best way of managing remote state, with features like caching, automatic re-fetching, pre-fetching, offline support, etc. Alternatives are SWR and RTK Query, but this is the most popular
UI State management	 <b>Context API</b>	There is almost no UI state needed in this app, so one simple context with <code>useState</code> will be enough. No need for Redux
Form management	 <b>React Hook Form</b>	Handling bigger forms can be a lot of work, such as manual state creation and error handling. A library can simplify all this
Other tools	React icons / React hot toast / Recharts / date-fns / Supabase	

STEP 4

- They have nothing right now, so they also need the API



→ to check in guests as they arrive.

### ❖ Development Phase I - Technology Decision:

Activities: Develop the API to handle user authentication, cabin and booking management.

### ★ HOW TO PLAN A REACT APPLICATION :

1. Gather application requirements and features.
2. Divide the application into pages.
3. Divide the application into feature categories.
4. Decide on what libraries to use (Technology Decisions).

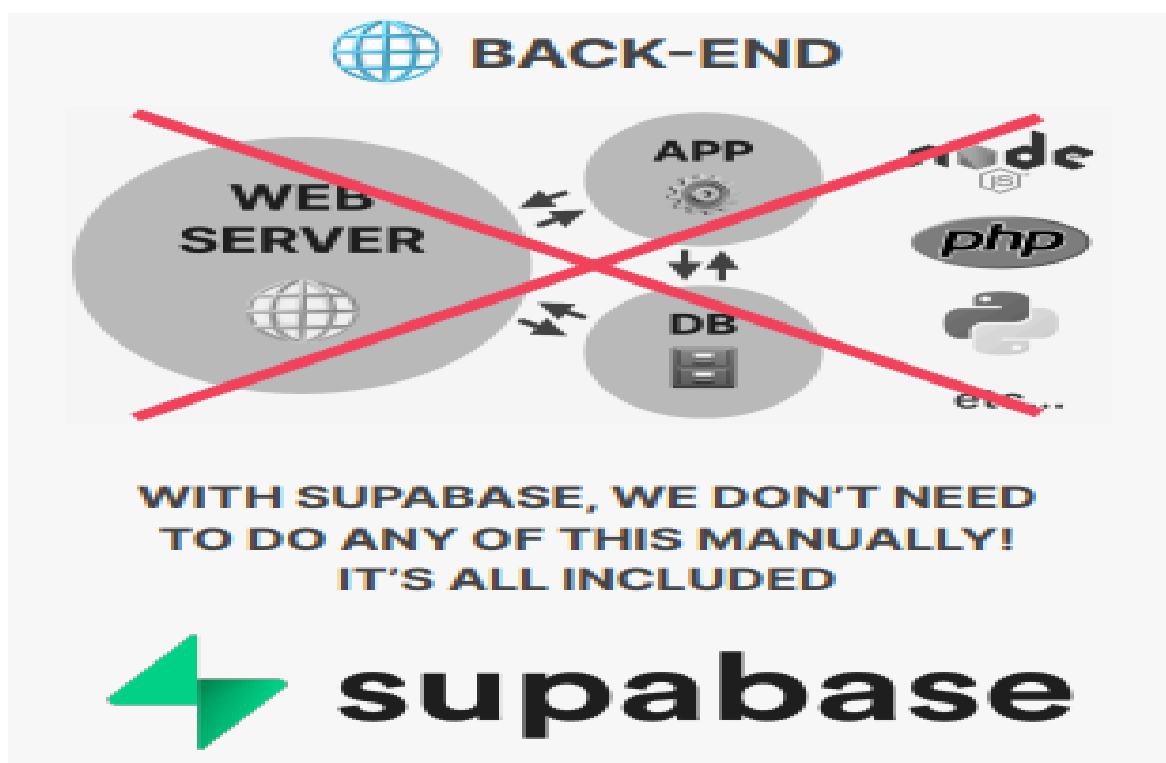


## ❖ Development Phase II - Backend API:

Activities: Develop the API to handle user authentication, cabin and booking management.



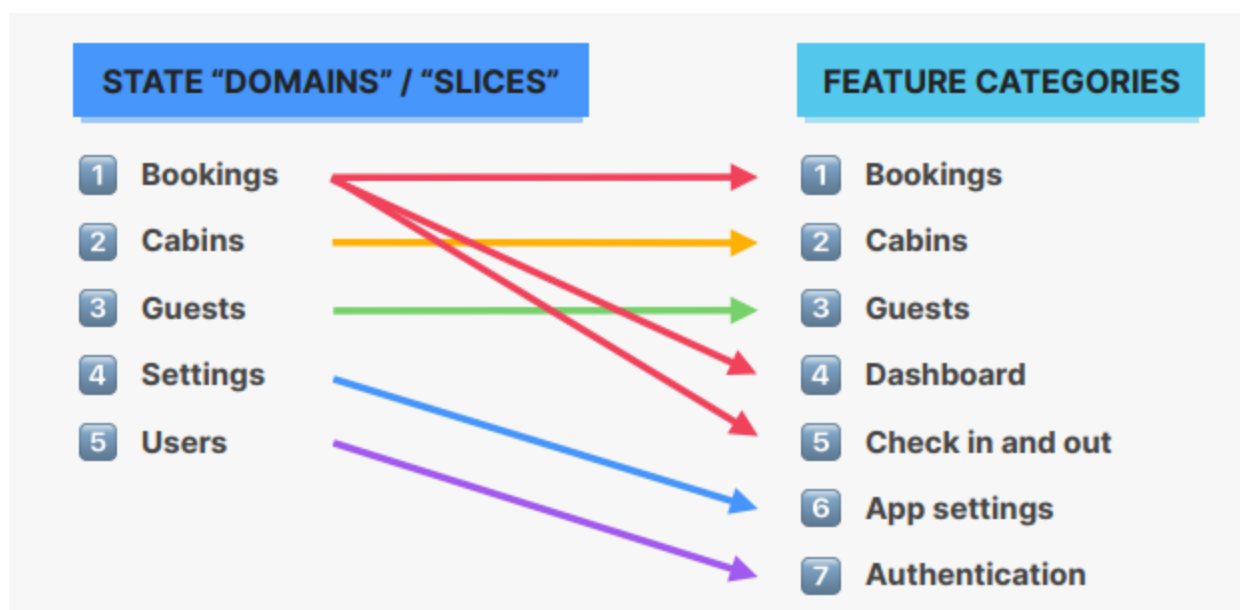
We need a service that allows developers to easily create a **database and API** which means **no back-end development** is needed so it is perfect to get up and running **quickly** → **Supabase** → Automatically creates a back-end with a Postgres database so we can easily request and receive data from the server.



So, to figure out which tables we should actually create inside Supabase, in our database, we first need to think about our **application state**(because it is a **Reactjs app**).

In a big application like this, the modeling state becomes a bit different because we think more about state categories or state slices on a much higher level.

So, we're not gonna plan state at a component level but really on the page and on the application feature level, so in order to power all these features (build them):



- Colors mean different states.
- The dashboard will simply display a few statistics about recent bookings and the check-in features are basically all about updating the booking from checked out to checked in or also the other way around for check out.
- There will be one table for each state "slice" in the database.

=====

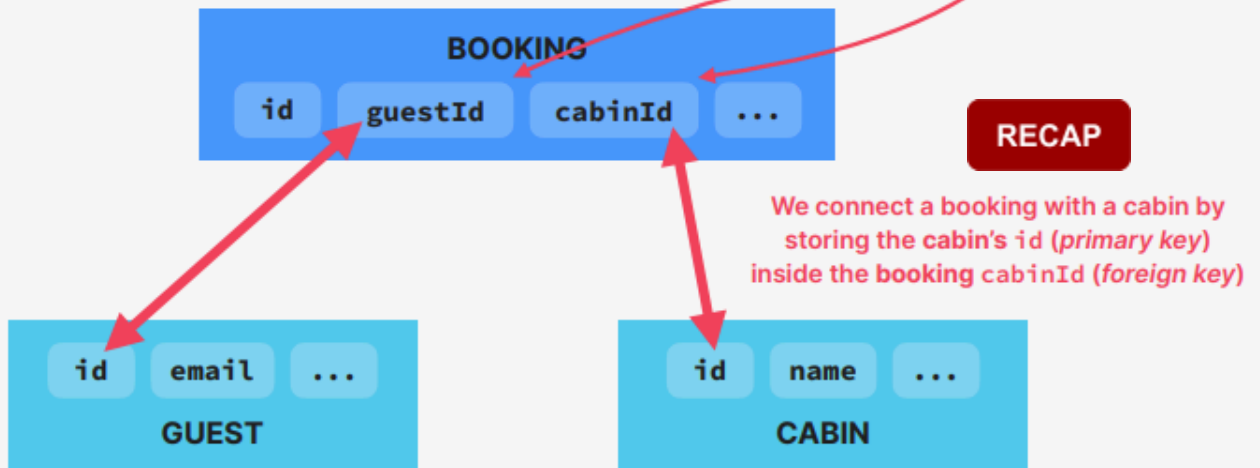
- ❖ All of this state will be a global remote state, stored within [Supabase](#) and managed on the front end using [React Query](#).

## MORE DETAILS:

👉 **Bookings** are about a **guest** renting a **cabin**

👉 So a booking needs information about what **guest** is booking which **cabin**: we need to **connect** them

👉 Supabase uses a Postgres DB, which is SQL (relational DB). So we **join** tables using **foreign keys**



- Each booking (**cabin's ID**) needs to know which guest is actually doing the booking(**booking's ID**).

### SO WE NEED TO:

1. Take the guest's ID and store that inside a new field inside the **\*booking** called guest ID.
2. The same thing with the cabin.
3. So, the bookings need to know which cabin was actually booked.
4. And so again, we will store the cabin's ID inside a special field of the booking called cabin ID.

**\* → Table Name**

## ❖ Relationships Between Tables:

### Foreign Key Relation



Table Editor

3. Select a column from cabins table to reference to Which will be id

Create a new table under public

Add foreign key relationship to bookings

What are foreign keys?

Select a schema: public

Select a table to reference to: cabins

Select columns from public.cabins to reference to: cabinid → id

Add another column

Column types will be updated  
The following columns will have their types updated to match their referenced column:  
cabinid → int8

Which action is most appropriate?

Cancel

Table Editor

4. Now this table is connected to cabinid

Create a new table under public

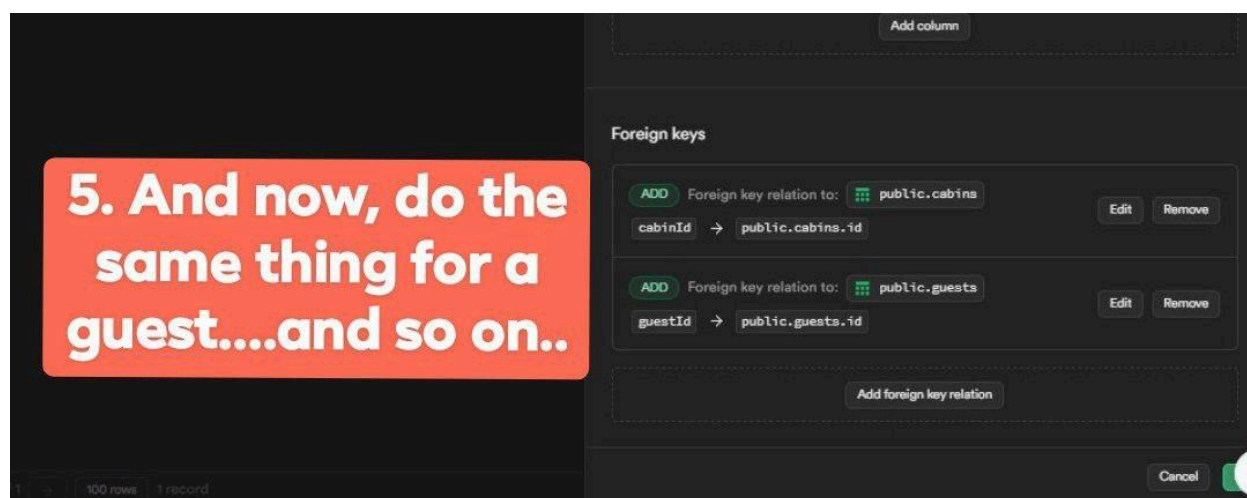
numnights: int2, numGuests: int2, cabinPrice: float4, extrasPrice: float4, totalPrice: int4, status: text, hasBreakfast: bool, isPaid: bool, observations: text, cabinid: int8

Column type cannot be changed as it has a foreign key relation

Foreign keys

ADD Foreign key relation to: public.cabins

Edit Remove Cancel



## NOTE:

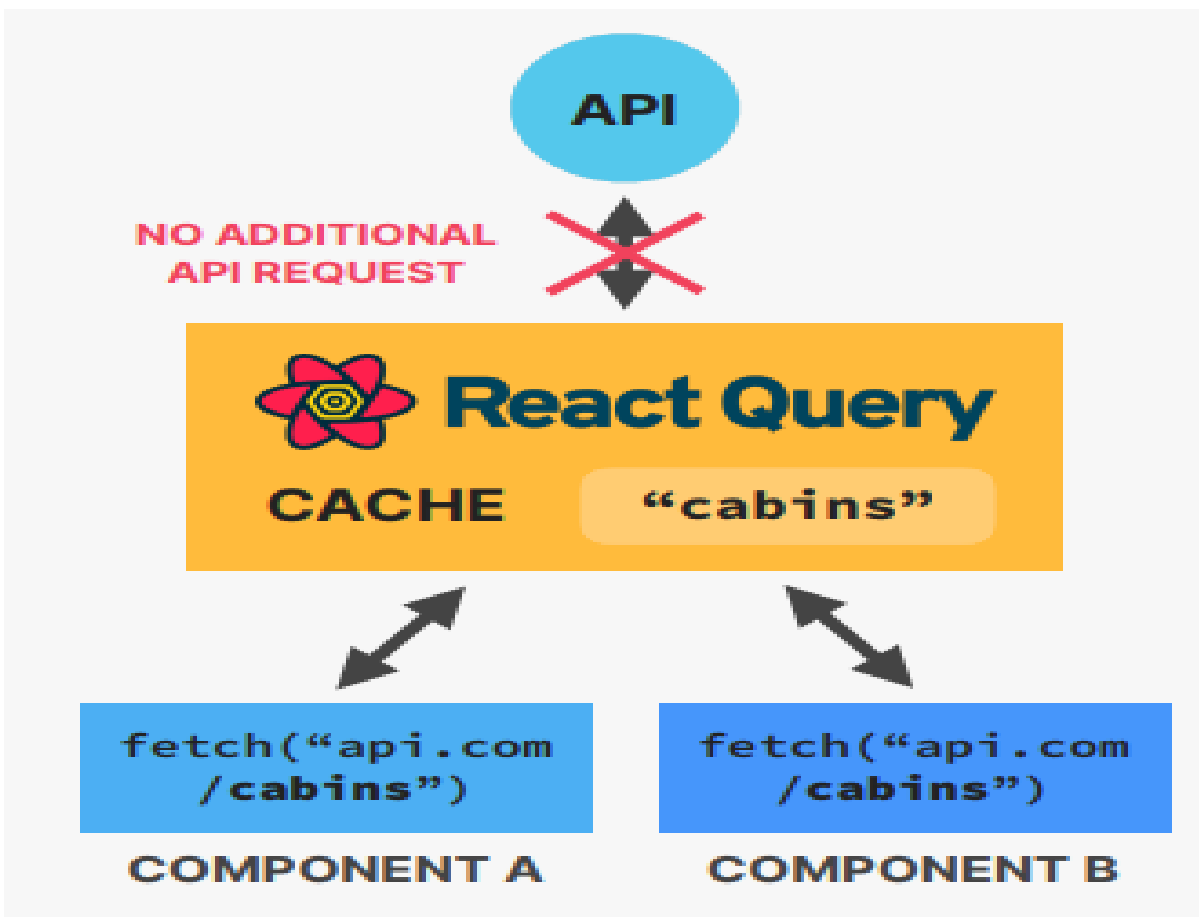
A foreign key relation in database management is a key used to link two tables together. A foreign key in one table points to a primary key in another table.

The purpose of the foreign key is to ensure the referential integrity of the data, meaning it restricts the data to only allow values that appear in the set of values in the linked table.

- ★ **Foreign Key:** This is a column or a set of columns in one table that uniquely identifies a row of another table or the same table. The foreign key matches the primary key field of another table to establish a relationship between the two.
- ★ **Primary Key:** This is a column, or a set of columns, in a table that uniquely identifies a row in that table. Each table should have a primary key, and each table can have only one primary key.
- ★ **Referential Integrity:** This is a database concept that ensures that relationships between tables remain consistent. When one table has a foreign key that points to another table, you can only enter values in that foreign key that exist as primary key values in the linked table unless null is permitted.



- Powerful library for managing remote (server) state.
- It contains many features that allow us to write a lot less code, while also making the UX a lot better:
  - Data is stored in a cache: which means that the fetched data will be stored in order to be used in different points of the applications.
  - Automatic loading and error states.
  - Automatic re-fetching to keep state synched.
  - Pre-fetching.
  - Easy remote state mutation (updating).
  - Offline support.



## ❖ Development Phase II - Frontend Application:

