# AIR-BnB Clone Docs

## Flow of Working with the files

**MAP AND CARD VIEW**

### 1-Result.jsx :

**Description:**

The Result component is responsible for displaying search results in either map view or card view based on user preference. It allows users to toggle between these views and provides additional functionalities such as filtering a State:

**State:**

- view: Represents the current view mode, either "map" or "card". This state variable is managed using the useState hook. Initially set to "card" view.
- value: Represents the current tab value for card view. This state variable is managed using the useState hook. Initially set to 0 and selecting search results.
- The useResponsive hook is utilized for determining if the current device is a desktop, which is used for responsive design purposes.

The component can be further extended or customized based on specific project requirements.

### 2-CardView.jsx :

**Description:**

The CardView component is a part of the user interface for displaying property listings in a tabular format. It provides users with a convenient way to navigate through different categories of properties.

**Props:**

view (string): Indicates the current view mode, which can be either "map" or "card".

value (number): Represents the index of the currently active tab.

handleChangeTab (function): Callback function invoked when a tab is changed.

The CardView component renders a set of tabs, each corresponding to a specific category of properties. Users can click on tabs to switch between categories and view property listings within each category. It offers a seamless navigation experience and enhances user engagement by organizing properties into logical groups.

**State:**

•**activeTab (number):** Tracks the index of the currently active tab. When the user selects a tab, this state is updated to reflect the new active tab index.

**Initial State:**

•**activeTab:** Initialized with a default value of 0, indicating the first tab is initially active.

**Purpose of State:** The activeTab state is crucial for managing the active tab in the CardView component. It ensures that the correct tab is highlighted and its corresponding properties are displayed to the user.

## 2.1 CustomTabPanel Component

The CustomTabPanel component is a reusable tab panel that conditionally renders its children based on the active tab index.

**Props:**

•index (number): Represents the index of the tab panel.

•value (number): Indicates the index of the active tab.

•children (ReactNode): The content to be rendered within the tab panel.

The CustomTabPanel component is designed to work seamlessly with the CardView component. It dynamically displays the content of each tab panel based on the active tab index, ensuring that only the content of the currently selected tab is visible to the user.

**PropTypes**:

The PropTypes section specifies the expected data types for the props passed to the CardView component, ensuring type safety and facilitating code maintenance.

•value (number): Specifies the index of the active tab.

•handleChangeTab (function): Defines the callback function triggered when a tab is changed.

•view (string): Indicates the current view mode, either "map" or "card".

## 3.0 Properites.jsx

**Properties Component**

The Properties component is responsible for rendering property listings within a grid layout. It displays essential details about each property, such as its image, price, location, and amenities.

**Props:**

•view (string): Indicates the current view mode, which can be either "map" or "card".

The Properties component generates a grid of property cards based on the provided data. Each card represents a property listing and contains relevant information for users to make informed decisions. It enhances the user experience by presenting property details in a visually appealing and easily accessible format.

**Images:**

•The images array contains URLs of property images used for rendering property thumbnails.

**Card Rendering:**

•The component dynamically generates property cards based on the provided data and the current view mode.

•Each card includes an image, rating, price, location, and amenities information.

**Interaction:**

•Users can interact with property cards to view more details or take further actions.

**PropTypes:**

•The view prop is required and must be either "map" or "card". It specifies the current view mode of the properties.

### 3.1 CardMedia:

Displays the property image as a background image within the card.

**Chip:**

•Shows the property rating as a chip overlay on the card image.

**IconButton:**

•Allows users to add properties to their favorites list.

**CardContent:**

•Contains detailed information about the property, including price, location, and amenities.

**Typography:**

•Displays textual information such as price, location, and amenity details.

**Stack:**

•Organizes content within the card layout, ensuring proper alignment and spacing.

**Divider:**

•Separates different sections of property details for better visual clarity.

**Iconography:**

•Icons such as Bed, Bathtub, and MapPin are used to represent property amenities and location.

**PropTypes:**

•The view prop is defined using PropTypes to ensure type safety and validation.

***Different properties componets are made in the code for different images as mapboxapi is unaccessable currently.***

## 4.0 Map.jsx

(caution=> MapBoxApi is currently not accessable)

**Map Component**

The Map component is responsible for rendering an embedded Google Maps iframe within a styled Card.

**Description:** The Map component integrates a Google Maps iframe to display a map of a specific location. It provides users with a visual representation of the geographical area and allows them to interact with the map features.

**Styling:**

•The MapWrapperStyle styled component defines custom styles for the map container.

•It sets the height, overflow, position, and border-radius properties to ensure proper rendering and aesthetics.

**iframe:**

•The iframe element loads the Google Maps embed URL with specific parameters to display the desired location.

•It includes attributes such as title, width, height, loading, and allowFullScreen to control iframe behavior and appearance.

**Card:**

•Wraps the map content within a Card component for better organization and presentation.

**CardContent:**

•Contains the map content, including the styled map wrapper and the embedded iframe.

**Google Maps Embed:**

•The Google Maps embed URL specifies the location (Toronto, Canada) and provides additional parameters for customization.

•Users can interact with the embedded map, zoom in/out, and view different map layers as per their preferences.

**PropTypes:**

•The Map component does not accept any props and does not require PropTypes validation.

# ***FILTER SECTION***

## 5.0 Filter Component

**Description:**

The Filter component is responsible for rendering various filtering options for refining search results.The Filter component provides users with options to filter search results based on different criteria such as type of place, price range, rooms and beds, and booking options. It enhances user experience by allowing them to customize their search parameters to find relevant listings.

**Structure:**

•The Filter component is wrapped within a Box component to provide padding and ensure proper spacing.

•It contains a Card component to visually group the filtering options together.

•The Card includes a header section with the title "Filters" and a "Clear All" button for resetting the filters.

Filtering Options:

**1.Type of Place:**

•Renders the TypeOfPlace component, which allows users to select the type of accommodation they are looking for (e.g., villa, castle, beach, etc.).

•The TypeOfPlace component provides a list of options for users to choose from.

**2.Price Range:**

•Displays the PriceRange component, enabling users to specify their preferred price range for accommodations.

•Users can input their desired minimum and maximum prices to filter search results accordingly.

**3.Rooms and Beds:**

•Includes the RoomsAndBeds component, which allows users to select the number of rooms and beds they require in the accommodation.

•Users can specify the desired number of bedrooms and beds to narrow down their search results.

**4.Booking Options:**

•Shows the BookingOptions component, providing users with additional booking-related filters such as check-in and check-out dates.

•Users can input their preferred dates to filter accommodations based on availability.

**Divider:**

•Utilizes the Divider component from MUI to create visual separation between different filtering sections.

•Ensures clear distinction and organization of filtering options for better user experience.

**Button:**

•The "Clear All" button allows users to reset all the selected filtering options and start over.

•Clicking the button clears all selected filters, providing users with a convenient way to refine their search criteria.

## 6.0 TypeOfPlace Component

**Description:**

The TypeOfPlace component is responsible for rendering options for selecting the type of accommodation.This component presents users with a radio button group allowing them to specify their preference for the type of accommodation they are looking for. It enhances the search experience by enabling users to filter search results based on their desired accommodation type.

**Structure:**

•The TypeOfPlace component is structured as a Stack component to ensure proper spacing and layout.

•It includes an InputLabel component to provide a label for the radio button group.

•The radio button group is implemented using the RadioGroup component, allowing users to select from multiple options.

•Each option is represented by a FormControlLabel component containing a Radio button and a corresponding label.

**Radio Button Group:**

•The radio button group is displayed horizontally (row layout) for better presentation.

•It has a default value of "any-type" to indicate that any type of accommodation is acceptable by default.

•Users can select from three options: "Any Type", "Room", and "Entire Home".

**Typography:**

•Typography variants are used to style the labels of the radio buttons for consistent and visually appealing presentation.

•Each label is styled with a specific font size and font weight for better readability and visual hierarchy.

**InputLabel:**

•The InputLabel component provides a label for the radio button group, indicating its purpose ("Type of Label").

•It ensures accessibility and clarity by describing the purpose of the radio button group to users.

## 7.0 PriceRange Component

Description: The PriceRange component allows users to specify a price range using a slider and input fields.This component provides users with a convenient interface to set their desired price range for accommodation search. It consists of a slider control for selecting a range visually and input fields for entering precise minimum and maximum price values.

**State:**

•The PriceRange component uses the useState hook to manage the state of the price range.

•It initializes the state with an array containing default minimum and maximum price values ([20, 75]).

•The state is updated using the setValue function whenever the user interacts with the slider or input fields.

**Slider:**

•The slider control allows users to visually select a price range by dragging a handle along a track.

•It displays the current selected range with automatic value label display (valueLabelDisplay="auto").

•The slider's value is controlled by the value prop, which is bound to the value state.

•When the user changes the slider position, the handleChange function is called to update the state with the new values.

**Input Fields:**

•Two text input fields are provided for entering precise minimum and maximum price values.

•The input fields are synchronized with the slider's value to ensure consistency.

•Each input field has a type="number" attribute to enforce numeric input.

•The value of each input field is bound to the corresponding value in the state array (value[0] for minimum and value[1] for maximum).

•When the user enters a new value in either input field, the respective value in the state array is updated accordingly.

**InputLabel:**

•An InputLabel component is used to provide a label for the price range section, enhancing clarity and accessibility.

•It describes the purpose of the input fields to users ("Price Range").

**Stack Layout:**

•The component utilizes the Stack component for proper spacing and layout management.

•Inputs are organized vertically in a Stack with appropriate spacing between them.

**PropTypes**:

•The PriceRange component does not accept any props and does not require PropTypes validation.

## 8.0 RoomsAndBeds Component

**Description:**

The RoomsAndBeds component allows users to select the number of bedrooms, beds, and halls using toggle buttons.This component provides users with a user-friendly interface to specify the number of bedrooms, beds, and halls for their accommodation search. It consists of multiple sections, each containing toggle button groups for selecting the desired quantity.

**State:**

The RoomsAndBeds component uses the useState hook to manage the state of the selected number of bedrooms, beds, and halls.

It initializes the bedroom state with the default value of '1'. The state is updated using the setBedroom function whenever the user interacts with the toggle buttons.

**Toggle Button Groups:**

I. Three toggle button groups are provided for selecting the number of bedrooms, beds, and halls.
II. Each toggle button group allows users to choose from multiple options using toggle buttons.
III. The value prop of each toggle button group is bound to the bedroom state to reflect the selected quantity.
IV. When the user selects a different quantity by clicking on a toggle button, the handleChangeBedroom function is called to update the state with the new value.

**Typography and InputLabel:**

**Typography components** are used to display labels for each section, indicating the type of room or area being selected.

**The InputLabel** component provides a descriptive label for the entire section, enhancing clarity and accessibility.

**Stack Layout:**

The component utilizes the Stack component for proper spacing and layout management.

Each section is organized within a Stack with appropriate spacing between them.

## 9.0 BookingOptions Component

**Description:**

The BookingOptions component provides users with options related to booking preferences, such as instant booking and self-check-in.This component allows users to specify their booking preferences for accommodation. It consists of toggles for instant booking and self-check-in, providing users with flexibility and control over their booking process.

**State:**

•The BookingOptions component does not utilize any local state. It only relies on the state management provided by the Switch components.

**Switch Toggles:**

•Two switch toggles are provided for instant booking and self-check-in options.

•Each toggle is represented by a Switch component from Material-UI.

•The defaultChecked attribute is used to set the initial state of the "Instant Book" toggle to checked.

•Users can toggle the switches on or off to indicate their preferences.

**InputLabel:**

•An InputLabel component is used to provide a label for the booking options section, enhancing clarity and accessibility.

•It describes the purpose of the booking options to users ("Booking Options").

# ***HEADER SECTION***

## 10.0 MainLayout Component

The MainLayout component serves as the main layout for the application, providing a structured layout that includes a header and content area.

**Description:**

The MainLayout component is responsible for rendering the overall layout structure of the application. It includes a header component at the top and a content area where nested routes are rendered using the Outlet component from react-router-dom.

### 10.1 Header Component:

•The Header component is imported and included at the top of the layout.

The Header component represents the header section of the application, containing elements such as a logo, search input field, and user avatar.

**Description**: The Header component is responsible for rendering the header section of the application. It includes elements such as a logo, search input field, and user avatar for user interaction and navigation.

### 10.2 Logo Component:

•The Logo component is imported and included within the header.

•It represents the logo or branding of the application.

•The Logo component is expected to be defined elsewhere in the project.

**10.3 Input Component:**

•The Input component is imported and included within the header.

•It represents a search input field for users to input search queries.

•The Input component is expected to be defined elsewhere in the project.

**10.4 Avatar Component:**

•The Avatar component from Material-UI is used to display a user avatar.

•The alt prop is set to a random name generated using the faker library to provide alternative text for accessibility.

•The src prop is set to a randomly generated avatar image URL using the faker library for demonstration purposes.

## 11.0 Input Component DateKeeper

The Input component represents the search input section of the application, allowing users to input search queries, select destination, check-in and check-out dates, and specify the number of guests for their accommodation.

**Description:** The Input component is responsible for rendering the search input section of the application. It includes elements such as destination search, date picker for check-in and check-out dates, and guest selector.

**State:**

•The Input component utilizes local state to manage the selected date range for check-in and check-out dates.

•value: State variable to hold the selected date range.

•setValue: Function to update the selected date range.

**Styled Components:**

•StyledIconButton: Custom styled IconButton component used for the search button. It inherits styles from the IconButton component and customizes the background color on hover.

**Hooks:**

•useResponsive: Custom hook imported from the 'useResponsive' module. It is used to determine whether the current device is mobile or not.

•useState: Hook imported from React for managing local component state.

**Dependencies:**

•dayjs: Library for parsing, validating, manipulating, and formatting dates.

•@mui/x-date-pickers-pro: Material-UI extension package for advanced date picker components.

**Child Components:**

•LocationSearch: Component for searching and selecting destination locations.

•DateRangePicker: Component for selecting date range for check-in and check-out dates.

•MobileDatePicker: Component for selecting dates in a mobile-friendly format.

•GuestSelector: Component for selecting the number of guests.

**Shortcuts:**

•The shortcutItems array contains predefined date range shortcuts for quick selection. It includes options such as "This week" and "Last 7 days".

**Layout:**

•The Input component uses a responsive layout, adjusting its appearance based on the device's screen size.

•It organizes elements in a card layout with responsive grid and stack components for better alignment and spacing.

**Button Handling:**

•The component renders a search button with a magnifying glass icon. The button's behavior varies based on the device's screen size:

•On mobile devices, it renders a contained button with the magnifying glass icon and the text "Search".

•On larger screens, it renders an icon button with a custom styled IconButton component.

# ***SEARCH SECTION***

## 12.0 LocationSearch Component

A list of countries in json is used in the code

The LocationSearch component provides an autocomplete search input for selecting locations. It allows users to enter text and displays suggestions based on the input text.

**Description:** The LocationSearch component renders an Autocomplete component from Material-UI, providing users with a searchable input field for selecting locations. It fetches and displays a list of suggested locations as the user types, allowing them to choose from the available options.

**State:**

•The LocationSearch component utilizes local state to manage the input text, options, and loading state.

•inputText: State variable to hold the current input text entered by the user.

•setInputText: Function to update the input text state.

•options: State variable to hold the list of location options.

•setOptions: Function to update the options state.

•loading: State variable to indicate whether data is being fetched.

•setLoading: Function to update the loading state.

**Functions:**

•filterOptions: Function to filter the list of cities based on the input text entered by the user.

•fetchLocations: Function to fetch and update the list of location options asynchronously. It simulates a delay of 1 second before updating the options state.

**Dependencies:**

•Material-UI components:

•Autocomplete: Component for providing autocomplete functionality.

•CircularProgress: Component for displaying a circular loading indicator.

•TextField: Component for rendering input fields.

**Layout:**

•The LocationSearch component renders an Autocomplete component with a TextField as its input.

•It utilizes fullWidth and freeSolo props to allow the input to occupy the entire width and accept free-form input.

•The Autocomplete component dynamically displays loading indicator (CircularProgress) while fetching data.

**Props:**

•The LocationSearch component does not accept any props from its parent component.

**Data Source:**

•The component uses a simulated list of cities as its data source for demonstration purposes. It filters and updates the list of options based on the user's input text.

**Input Handling:**

•The component captures user input in the Autocomplete input field and triggers the onInputChange callback.

•As the input text changes, it updates the inputText state and fetches location options based on the entered text.

**Rendering:**

•The Autocomplete component renders a TextField input with various props and configurations.

•It customizes the input field by providing placeholder text, disabling underline, and displaying loading indicator if data is being fetched.

**Behavior:**

•As the user types in the input field, the component dynamically fetches and displays matching location options.

•It provides a smooth and responsive autocomplete experience, allowing users to quickly select desired locations.

# ***GUEST SECTION***

## 13.0 GuestSelector Component

The GuestSelector component allows users to select the number of adults, children, pets, and Boomers for accommodation purposes. It provides an interactive interface with buttons to increment or decrement the count for each category.

**Description:** The GuestSelector component renders a TextField input that displays the selected number of guests for each category. It also includes buttons to increment or decrement the guest count. Clicking on the input opens a popover with controls to adjust the guest count for each category.

**State:**

•The GuestSelector component utilizes local state to manage the popover anchor element (anchorEl) and the number of guests for each category (guests).

•anchorEl: State variable to store the anchor element for the popover.

•setAnchorEl: Function to update the anchor element state.

•guests: State variable to store the number of guests for adults, children, pets, and Boomers.

•setGuests: Function to update the guest count state.

**Functions:**

•handleClick: Function to handle click events on the TextField and set the anchor element for the popover.

•handleClose: Function to close the popover.

•handleGuestChange: Function to handle changes in the guest count for each category (increment or decrement).

**Layout:**

•The GuestSelector component renders a TextField input that displays the selected number of guests.

•Clicking on the TextField opens a popover containing controls to adjust the guest count.

•The popover includes sections for adults, children, pets, and Boomers, each with buttons to increment or decrement the count.

**Behavior:**

•Clicking on the TextField opens a popover with controls to adjust the guest count.

•Users can use the plus and minus buttons to increment or decrement the count for each guest category.

•The guest count is displayed in the TextField, updating dynamically as users adjust the count.

**Dependencies:**

•Material-UI components:

•TextField: Component for rendering input fields.

•Typography: Component for displaying text.

•Popover: Component for displaying popovers.

•IconButton: Component for rendering icon buttons.

•Divider: Component for rendering dividers between sections.

•Phosphor Icons:

•PlusCircle: Icon representing addition.

•MinusCircle: Icon representing subtraction.

**Props:**

•The GuestSelector component does not accept any props from its parent component.

**Input Handling:**

•Users can adjust the guest count by clicking on the plus and minus buttons in the popover.

•The component dynamically updates the guest count state as users make changes.

**Rendering:**

•The GuestSelector component renders a TextField input to display the selected number of guests.

•Clicking on the input opens a popover containing controls to adjust the guest count.

## 15.0 References:

https://react.dev/

https://mui.com/material-ui/

https://developers.google.com/maps

https://www.mapbox.com/install/javascript/bundler-install

# 16.0 Challenges and Solutions :

**1.0 Challenge:** In the stack component, the children were not being properly aligned due to styling issues. This resulted in inconsistent layout and appearance.

**1.0 Solution:** To address this issue, a grid component was created with both container and item properties. This allowed for proper alignment and styling of the children elements within the stack component, ensuring a consistent and visually pleasing layout.

**2.0 Problem Encountered**: While working on the Airbnb clone frontend project, there was an issue where clicking on each tab (e.g., Castle, Park, Villa) should render images associated with each tab. However, only one tab was rendering images as expected, while the others remained inactive or did not display the corresponding content.

**2.0 Challenge Description:** The challenge stemmed from the lack of synchronization between the tabs and their associated content rendering. Without proper management of tab activation states, only one tab was being recognized as active, leading to the incorrect rendering of images associated with that particular tab. As a result, the user experience was compromised, as they expected to see different images based on the selected tab but were only presented with images from one tab.

**2.0 Resolution:** To address this challenge, an "isActive" state variable was introduced to manage the active state of each tab. By maintaining the active state for each tab separately, clicking on a tab would correctly render the associated images, ensuring that the user could view different sets of images based on the selected tab. Implementing the "isActive" state allowed for proper synchronization between the tabs and their corresponding content, enhancing the user experience by providing the expected behavior of tab-based image rendering.

**3.0 Challenge:**

The Airbnb clone frontend project encountered issues due to the requirement for a valid Mapbox access token. Without it, Mapbox-GL services couldn't render maps, leading to a deficient user interface.

**3.0 Solution:** To resolve this, the project implemented prompts for a valid Mapbox access token and themes, ensuring seamless integration of Mapbox features and enhancing user customization options. (caution=>mapboxapi is currently unaccessable)

**4.0 Challenge:** In the guest selection component, when clicking on the plus button to increase the count for one guest type (e.g., pets), the counts for all guest types (e.g., adults, children, boomers) were increasing simultaneously. This behavior is unintended and leads to incorrect guest count adjustments.

**4.0 Solution:** To resolve this issue, ensure that the event handler for incrementing the count of a specific guest type only affects that particular guest type's count state. Verify that the event handler function handleGuestChange correctly updates the state for the targeted guest type based on the button clicked (plus or minus) and only modifies the count for that specific guest type. Double-check the implementation to confirm that the correct state property is being updated within the function.