Table of contents

Byanat

To run the project, you have to run:

```
npm install
```

And after that, run:

```
npm run dev
```

And to test the app, run:

```
npx cypress open
```

Search Box Component

The SearchBox component provides a user interface for selecting filters and searching for cities. It comprises two main sections: a dropdown for selecting filters and an autocomplete input for searching cities.

Imports

```
import { Dropdown } from 'primereact/dropdown'
import { useState } from 'react'
import {
    AutoComplete,
    AutoCompleteCompleteEvent,
} from 'primereact/autocomplete'
import { Button } from 'primereact/button'
import { SearchIcon } from '../../../assets/icons'
import { City } from '../../../assets/types'
import { useDispatch, useSelector } from 'react-redux'
import { AppDispatch, RootState } from '../../../store'
import { setFilter } from '../../../store/slices/filterSlice'
import { setSelectedCity } from '../../../store/slices/citySlice'
```

DropdownSection Component

The DropdownSection component renders a dropdown menu for selecting filters.

```
const DropdownSection = () => {
    const dispatch = useDispatch<AppDispatch>()
    const selectedFilter = useSelector(
        (state: RootState) => state.filter.selectedFilter
    )

    const groupedFilters = [
        {
            label: 'Type',
            items: [
                {
                    label: 'Entire Studio Apartment',
                    value: 'Entire Studio Apartment',
                },
                { label: 'Entire Home', value: 'Entire Home' },
                {
                    label: 'Share with Super Host',
                    value: 'Share with Super Host',
                },
            ],
        },
        {
            label: 'Rating',
            items: [
                { label: 'Less than 3', value: '3' },
                { label: 'Between 3 and 4', value: '4' },
                { label: 'More than 4', value: '5' },
            ],
        },
    ]

    const groupedItemTemplate = (option: { label: string }) => (
        <div className="align-items-center flex">
            <div>{option.label}</div>
        </div>
    )

    const handleChange = (e: { value: string }) => {
        dispatch(setFilter(e.value))
    }

    return (
        <Dropdown
            value={selectedFilter}
            onChange={handleChange}
            options={groupedFilters}
            optionLabel="label"
            optionGroupLabel="label"
```

```
                optionGroupChildren="items"
                optionGroupTemplate={groupedItemTemplate}
                className="max-w-[150px] md:w-[250px]"
            />
        )
}
```

- **selectedFilter**: The currently selected filter, retrieved from the Redux store.
- **groupedFilters**: The filter options grouped by categories such as "Type" and "Rating".
- **groupedItemTemplate**: Template for rendering each grouped item.
- **handleChange**: Dispatches the selected filter to the Redux store.

SearchSection Component

The `SearchSection` component provides an autocomplete input for searching and selecting cities.

```
const SearchSection = () => {
    const dispatch = useDispatch()
    const selectedCity = useSelector(
        (state: RootState) => state.city.selectedCity
    )

    const cities: City[] = [
        { name: 'Dubai', code: 'DXB' },
        { name: 'Muscat', code: 'MSC' },
        { name: 'Tehran', code: 'TEH' },
    ]

    const [filteredCities, setFilteredCities] = useState<City[] |
undefined>(
        undefined
    )

    const search = (event: AutoCompleteCompleteEvent) => {
        let _filteredCountries: City[]
        if (!event.query.trim().length) {
            _filteredCountries = [...cities]
        } else {
            _filteredCountries = cities.filter((city) =>

city.name.toLowerCase().includes(event.query.toLowerCase())
            )
        }

        setFilteredCities(_filteredCountries)
    }

    const handleCityChange = (e: { value: City[] }) => {
        if (e.value.length > 0) {
            const lastSelectedCity = e.value[e.value.length - 1]
            dispatch(setSelectedCity([lastSelectedCity]))
        }
    }

    return (
        <AutoComplete
            field="name"
            multiple
            value={selectedCity}
            suggestions={filteredCities}
            completeMethod={search}
            onChange={handleCityChange}
            className="w-full min-w-full"
            pt={{ root: { overflow: 'scroll', width: '100%' } }}
        />
    )
}
```

- **selectedCity**: The currently selected city, retrieved from the Redux store.
- **cities**: List of available cities to search.

- **filteredCities**: The list of cities filtered based on the search query.
- **search**: Filters the cities based on the user's search query.
- **handleCityChange**: Dispatches the selected city to the Redux store.

Main Component

The SearchBox component combines the DropdownSection and SearchSection components and adds a search button.

```
export default function SearchBox() {
    return (
        <div className="cy-searchbox flex h-14 w-full lg:w-
[600px]">
            <div className="flex w-full rounded-l-md border-y
border-l border-slate-300">
                <DropdownSection />
                <SearchSection />
            </div>
            <Button
                aria-label="Search"
                className="flex h-14 w-14 items-center justify-
center rounded-l-none bg-[#5E81F4]"
            >
                <SearchIcon />
            </Button>
        </div>
    )
}
```

- Combines the DropdownSection and SearchSection components.
- Includes a search button with an icon.

Redux Integration

This component relies on Redux for state management. Ensure that the `filterSlice` and `citySlice` are properly set up in your Redux store.

Dependencies

- `primereact/dropdown`
- `primereact/autocomplete`
- `primereact/button`
- Redux setup with `react-redux`

Widget Container Component

The `WidgetContainer` component renders a list of draggable and resizable widgets in a vertical pane.

Imports

```
import { ReactNode, useCallback } from 'react'
import WidgetCard from '../WidgetCard'
import { Pane, SortablePane } from 'react-sortable-pane'
```

Interface

```
interface Item {
    id: number
    body: ReactNode
    title?: string
    subtitle?: string
}
```

- **Item**: Defines the structure of each card.
  - `id`: Unique identifier for the card (number).
  - `body`: Content of the card (ReactNode).
  - `title` (optional): Title of the card (string).
  - `subtitle` (optional): Subtitle of the card (string).

Initial Cards

```
const initialCards: Item[] = [
    {
        id: 1,
        body: <div>hello</div>,
        title: 'P&L',
        subtitle: 'Total profit growth of 25%',
    },
    {
        id: 2,
        body: <div>hello</div>,
        title: 'Current Plan',
        subtitle: 'Information and usages of your current plan',
    },
    {
        id: 3,
        body: <div>hello</div>,
    },
]
```

- **initialCards**: Array of card objects to be rendered initially.

Main Component

```
export default function WidgetContainer() {
    const renderCard = useCallback((card: Item) => {
        return (
            <Pane
                key={card.id}
                defaultSize={{ width: '100%', height: '32%' }}
                resizable={{ x: false, y: true, xy: false }}
            >
                <WidgetCard title={card.title} subtitle=
{card.subtitle}>
                    {card.body}
                </WidgetCard>
            </Pane>
        )
    }, [])

    return (
        <SortablePane direction="vertical" className="w-full"
margin={16}>
                {initialCards.map((card) => renderCard(card))}
            </SortablePane>
    )
}
```

- **WidgetContainer**: Main component rendering the sortable pane with the cards.
    - Uses `useCallback` to memoize the card rendering logic.
    - Maps over `initialCards` to render each card using the `renderCard` function.
- **Items**: Each item contains an `id`, a `body` (ReactNode), and optionally a `title` and `subtitle`.
- **SortablePane**: Used to create a container that allows sorting of the cards vertically.
- **Pane**: A wrapper for each card, which is resizable vertically.

Map Component

Overview

MapComponent is a React component that integrates with Mapbox to display a map with interactive features. It allows users to view markers and popups for specific locations, fetch geoJSON data, and interact with map elements.

Imports

```
import mapboxgl, { Map } from 'mapbox-gl'
import { useCallback, useEffect, useRef, useState } from 'react'
import 'mapbox-gl/dist/mapbox-gl.css'
import { FeatureProperties, GeoJSONResponse } from
'../../assets/types'
import { useDispatch, useSelector } from 'react-redux'
import { setGeoJSON } from '../../store/slices/geojsonSlice'
import { RootState } from '../../store'
import HoverCard from './HoverCard'
import { createRoot } from 'react-dom/client'
import { setHotel } from '../../store/slices/hotelSlice.ts'
import { setNewHotel } from '../../store/slices/newHotelSlice.ts'
import NewHotel from '../Modal/NewHotel'
```

Constants

Access Token: Token for Mapbox API. Tile ID: Mapbox tile ID. Initial Coordinates: Default coordinates for specific cities.

```
const accessToken = 'YOUR_MAPBOX_ACCESS_TOKEN'
const tileID = 'YOUR_TILE_ID'

const initialCoordinates = {
    Muscat: [58.38, 23.58],
    Dubai: [55.27, 25.2],
    Tehran: [51.37, 35.74],
}
```

Fetch GeoJSON Data

Fetches geoJSON data for a given coordinate.

```
const fetchGeoJSON = async (center: [number, number]) => {
    const radius = 100000000
    const limit = 50
    const query = await fetch(

`https://api.mapbox.com/v4/${tileID}/tilequery/${center[0]},${cent
er[1]}.json?
radius=${radius}&limit=${limit}&access_token=${accessToken}`,
        { method: 'GET' }
    )
    return await query.json()
}
```

Main Component

```
export default function MapComponent() {
    const mapContainerRef = useRef<HTMLDivElement | null>(null)
    const mapRef = useRef<Map | null>(null)
    const [lng, setLng] = useState<number>
(initialCoordinates.Muscat[0])
    const [lat, setLat] = useState<number>
(initialCoordinates.Muscat[1])
    const [zoom, setZoom] = useState<number>(12)
    const city = useSelector((state: RootState) =>
state.city.selectedCity)
    const dispatch = useDispatch()

    useEffect(() => {
        if (city) {
            const coordinates = initialCoordinates[city[0].name]
            if (coordinates) {
                setLng(coordinates[0])
                setLat(coordinates[1])
            }
        }
    }, [city])
}
```

Fetch and Display GeoJSON Data

Fetches geoJSON data and updates the map.

```
const handleGeoJSONFetch = useCallback(
       async (center: [number, number]) => {
           try {
               const json: GeoJSONResponse = await
fetchGeoJSON(center)
               const geoJSON = {
                   type: 'FeatureCollection',
                   features: json.features.map((feature) => ({
                       type: 'Feature',
                       geometry: {
                           type: 'Point',
                           coordinates:
feature.geometry.coordinates,
                       },
                       properties: feature.properties,
                   })),
               }

               if (mapRef.current?.getSource('tilequery')) {

mapRef.current.getSource('tilequery').setData(geoJSON)
               }
               dispatch(setGeoJSON(geoJSON))

               geoJSON.features.forEach((feature) => {
                   const coordinates =
feature.geometry.coordinates
                   const properties = feature.properties as
FeatureProperties
                   const el = document.createElement('div')
                   el.className = 'marker'
                   el.style.backgroundColor = 'white'
                   el.style.border = '1px solid gray'
                   el.style.padding = '5px'
                   el.style.borderRadius = '5px'
                   el.innerHTML = `$${properties.PRICE}`
                   el.style.textAlign = 'center'
                   el.style.width = '50px'

                   new mapboxgl.Marker(el)
                       .setLngLat(coordinates)
                       .addTo(mapRef.current!)
               })
           } catch (error) {
               console.error('Error fetching tile query results:',
error)
           }
       },
       [dispatch, city]
   )
```

Add Tile Query Source and Layer

Configures the map to display the tile query layer.

```
const addTileQuerySourceAndLayer = useCallback(() => {
    if (!mapRef.current) return

    mapRef.current.addSource('tilequery', {
        type: 'geojson',
        data: {
            type: 'FeatureCollection',
            features: [],
        },
    })

    const popup = new mapboxgl.Popup()

    mapRef.current.on('mouseenter', 'tilequery-points', (event) =>
{
        const features = event.features
        if (features && features.length > 0) {
            mapRef.current!.getCanvas().style.cursor = 'pointer'
            const coordinates =
features[0].geometry.coordinates.slice()
            const properties = features[0].properties as
FeatureProperties
            popup
                .setLngLat(coordinates)
                .setDOMContent(
                    (() => {
                        const container =
document.createElement('div')
                        const root = createRoot(container)
                        root.render(
                            <HoverCard
                                bedroom={properties.BEDROOMS}
                                bathroom={properties.BATHROOMS}
                                price={properties.PRICE}
                            />
                        )
                        return container
                    })()
                )
                .addTo(mapRef.current!)
        }
    })

    mapRef.current.on('mouseleave', 'tilequery-points', () => {
        mapRef.current!.getCanvas().style.cursor = ''
        popup.remove()
    })
    mapRef.current.on('click', 'tilequery-points', (event) => {
        const features = event.features
        if (features && features.length > 0) {
            const properties = features[0].properties as
FeatureProperties
            dispatch(setHotel(properties))
        }
    })
}, [])
```

Initialize Map

Initializes the map and sets up event listeners.

```
const initializeMap = useCallback(() => {
    if (mapRef.current) return

    mapRef.current = new mapboxgl.Map({
        container: mapContainerRef.current!,
        style: 'mapbox://styles/mapbox/streets-v12',
        center: [lng, lat],
        zoom: zoom,
        accessToken,
    })

    mapRef.current.on('load', () => {
        addTileQuerySourceAndLayer()
        handleGeoJSONFetch([lng, lat])
    })

    mapRef.current.on('move', () => {
        const { lng, lat } = mapRef.current!.getCenter()
        setLng(Number(lng.toFixed(4)))
        setLat(Number(lat.toFixed(4)))
        setZoom(Number(mapRef.current!.getZoom().toFixed(2)))
    })
    mapRef.current!.on('style.load', () => {
        mapRef.current!.on('dblclick', (e) => {
            const coordinates = e.lngLat
            dispatch(
                setNewHotel({
                    latitude: coordinates.lat,
                    longitude: coordinates.lng,
                    ADDRESS_LINE1: '',
                    BATHROOMS: 0,
                    BEDROOMS: 0,
                    CITY: '',
                    COUNTRY: '',
                    GUESTS: 0,
                    HOTEL_NAME: '',
                    NBHD_NAME: '',
                    PRICE: 100,
                    RATING: 5,
                    TYPE: '',
                })
            )
        })
    })
}, [lng, lat, zoom, handleGeoJSONFetch,
addTileQuerySourceAndLayer])
```

Effect Hooks

Setup and update the map on component mount and city changes.

```
useEffect(() => {
    initializeMap()
}, [initializeMap])

useEffect(() => {
    if (mapRef.current && city) {
        mapRef.current.setCenter([lng, lat])
        handleGeoJSONFetch([lng, lat])
    }
}, [handleGeoJSONFetch, city])

return (
    <div className="cy-map h-full overflow-hidden rounded-xl">
        <div ref={mapContainerRef} className="h-full" />
        <NewHotel />
    </div>
)
```