App.js Routing Summary

The main application router that defines all page routes for your State Park application.

Route Configuration

Path	Component	Description
`/`	`Home`	Landing page (same as `/Home`)
`/Home`	`Home`	Main homepage
`/About-Us`	`AboutUs`	About us information page
`/Sign-Up`	`AuthPage`	Authentication page (sign-up/login)
`/Map`	`MapComponent`	Interactive park map with filtering
`/Reservation`	`ReservationForm`	Park reservation form
`/My-Reservation`	`MyReservationPage`	User's reservation management
`/report`	`ParkStatusReportForm`	Park status reporting form

Key Features

1. **React Router v6**
    - **Uses `BrowserRouter` for client-side routing**
    - **`exact` prop ensures precise route matching**
2. **Styling**
    - **Imports Bootstrap CSS (`bootstrap/dist/css/bootstrap.min.css`)**
    - **Includes custom `App.css` for additional styling**
3. **Authentication Flow**
    - **`/Sign-Up` route handles both sign-up and login via `AuthPage`**
4. **Park Management**
    - **Complete workflow from map viewing → reservations → status reporting**

MapComponent Summary

A React component that renders an interactive map (using **Leaflet**) displaying state parks with filtering, spotlight mode, and user location features.

Key Features

1. **Map Initialization**
    - Uses `Leaflet` to render a map with OpenStreetMap tiles.
    - Default center: `[30.43, -84.28]` (or user-provided `lat`/`lon` from URL params).
    - Auto-locates the user if permissions allow.
2. **Park Markers**
    - Fetches park data from Firebase (`getObjects`).
    - Dynamically assigns icons based on park status:
        - `openPark` (capacity ≤ 15), `halwayOpenPark` (≤ 23), `closedPark` (default).
    - Popups show park details (name, capacity, etc.) with buttons for:
        - **Reservation** (navigates to `/reservation` with park ID).
        - **Report** (navigates to `/report` for status updates).
        - **Spotlight** (highlights a single park).
3. **Spotlight Mode**
    - Toggleable to focus on one park (uses `spotlightIcon`).
    - Exclusively shows the selected park and user location.

4. **Radius Filter**
    - Slider (0–50 miles) adjusts a circular radius to filter visible parks.
    - Parks within the radius are listed in a sidebar (sorted by distance).
5. **User Location**
    - Marks the user's location with a `Personal` icon.
    - Falls back to URL-provided `lat`/`lon` if geolocation is denied.
6. **UI Elements**
    - **Toggleable Sidebar**: Lists filtered parks with quick actions.
    - **Back Button**: Returns to the previous page.
    - **Slider**: Controls the search radius.

Dependencies

- `leaflet` (for maps), `geolib` (distance calculations), `react-router-dom` (URL params).
- Custom icons (`openPark`, `closedPark`, etc.) and components (`Slider`, `ParkToggleButton`).

Key Functions

- `initializeAllMarkers()`: Renders park/user markers.
- `toggleSpotlight(park)`: Toggles spotlight mode.
- `clearAllMarkers()`: Removes markers before updates.
- `filterParksWithinRadius()`: Filters parks by slider value.

State Management

- `parks`/`filteredParks`: Store park data and filtered results.
- `userLocation`: Tracks geolocation.
- `isSpotlightMode`/`spotlightedPark`: Manages spotlight state.

Usage Notes

- Ensure Firebase (`getObjects`) and icon paths are correctly configured.
- Styled with inline CSS for positioning (e.g., absolute panels, z-index layers).

Authentication System Summary

A React-based authentication module with **email/password** and **Google sign-in**, integrated with Firebase.

Components Overview

1. `EmailSignIn`

- **Functionality**:
  - Toggles between **sign-up** and **sign-in** modes.
  - Validates email/password inputs.
  - Uses Firebase `createUserWithEmailAndPassword` (sign-up) or `signInWithEmailAndPassword` (sign-in).
  - Saves user data to Firebase Realtime Database (`users/{uid}`) on sign-up.
  - Redirects to `/Home` on success.
- **Error Handling**:
  - Displays popup errors for:
    - Empty fields.
    - Weak passwords (`auth/weak-password`).
    - Existing accounts (`auth/email-already-in-use`).
    - Invalid credentials (`auth/user-not-found`).
- **UI**:
  - Input fields for email/password.
  - Toggle link between "Sign up" and "Log in".

---

2. `GoogleSignIn`

- **Functionality**:
  - Triggers Google sign-in via `signInWithPopup`.
  - Saves user data (email, UID, role) to Firebase Database.
  - Redirects to `/Home` on success.
- **Error Handling**:
  - Displays popup for failed sign-in or database errors.
- **UI**:
  - Button labeled "Sign in with Google".

---

3. `LogoutButton`

- **Functionality**:
  - Calls Firebase `signOut` and redirects to `/` (homepage).
- **UI**:
  - Simple button labeled "Logout".

Shared Utility: `ErrorPopup`

- A reusable popup component to display error messages.
- **Props**:
    - `message`: Error text.
    - `onClose`: Clears the error state.

Key Firebase Methods Used

Method	Purpose
`createUserWithEmailAndPassword`	Registers new email/password users.
`signInWithEmailAndPassword`	Logs in existing users.
`signInWithPopup`	Handles Google OAuth sign-in.
`setPersistence`	Persists auth state (uses `browserLocalPersistence`).
`signOut`	Logs out the current user.
`set(ref(db, path))`	Writes user data to Firebase Realtime DB.

Dependencies

- **Firebase**: `auth`, `database` (configured in `firebase-config.js`).
- **Routing**: `react-router-dom` (for `navigate`).

Usage Notes

1. Ensure Firebase is initialized with correct configs (API keys, auth providers).
2. Database rules should allow writes to `users/{uid}`.
3. Styling can be enhanced (inline CSS used for simplicity).

Firebase Database Utilities & Object Management

1. Core Functions

Function	Purpose	Key Features
`ensureBooleanField()`	Validates/updates `isOpen` field for all objects	- Converts non-boolean `isOpen` to `false` - Logs update status
`addCapacityToAllObjects()`	Adds missing capacity fields to objects	- Sets random `capacity` (1-25) - Initializes `currentCapacity` to 0 - Handles `parkingCapacity` if numeric
`addObj(newObject)`	Adds a new object to Firebase	- Uses object name as key - Logs success/error

2. `Homer` Component

A form to submit new objects to Firebase.

State Management:

- Tracks form data (`id`, `name`, `county`, `size`, etc.) via `useState`.
- Resets to `initialFormState` after submission.

Features:

- **Form Fields**:
  plaintext
- ID, Name, County, Size, Year Established, Water Body, Remarks, Longitude, Latitude
- **Validation**: Requires all fields before submission.
- **Submission**:
  - Calls `addObj()` with `timestamp`.
  - Logs errors for incomplete forms.

UI Structure:

```
jsx
<div>
  <h1>Enter Object Details</h1>
  {Object.keys(initialFormState).map(field => (
    <div key={field}>
      <label>{field}:</label>
```

```
      <input
        type="text"
        name={field}
        value={formData[field]}
        onChange={handleChange}
      />
    </div>
  ))}
  <button onClick={handleSubmit}>Submit</button>
</div>
```

Key Firebase Methods Used

Method	Purpose	Example Usage
`ref()`	Creates database reference	`ref(database, 'objects/')`
`get()`	Fetches data	`get(objectsRef).then(snapshot => ...)`
`set()`	Writes data	`set(ref(database, 'objects/' + name), data)`
`update()`	Batch updates	`update(ref(database), { path: value })`

Error Handling

- **Logs**: Console logs success/failure for all operations.
- **Fallbacks**: Defaults `isOpen` to `false`, `currentCapacity` to `0`.

Usage Notes

1. **Database Structure**:
    - Objects stored under `objects/{name}` with fields like `capacity`, `isOpen`, etc.
2. **Dependencies**:
    - Firebase Realtime Database (`firebase/database`).
3. **Testing**:

- Call `ensureBooleanField()` or `addCapacityToAllObjects()` manually if needed.

MyReservationPage Component Summary

A React component that displays a user's park reservations by email, with authentication handling.

Key Features

1. **Reservation Fetching**
   - Retrieves reservations via `getReservationsByEmail(email)` from Firebase.
   - Accepts email:
     - From URL parameter (`?email=...`) for admin/shared access.
     - From authenticated user (via `onAuthStateChanged`).
     - Manual input (for unauthenticated queries).
2. **Authentication Flow**
   - Auto-populates email if user is logged in (`auth.currentUser`).
   - Shows email input field only for unauthenticated users without URL params.
3. **Data Presentation**
   - Formats dates with `toLocaleString()`.
   - Displays:
     - Park name
     - Reservation name
     - Date/time
     - Guest count
     - Phone number
4. **UI Elements**
   - Conditional rendering based on auth state.
   - "Home" button for navigation.

Error Handling

- Validates email input before fetching.
- Handles invalid dates gracefully.

Dependencies

- Firebase: `auth`, `getReservationsByEmail` (custom function).
- Routing: URL query params (`?email=`).

ReservationForm Component Summary

A React form for submitting park reservations to Firebase, with automatic user data population and capacity tracking.

Key Features

1. **Form Data Management**
    - **State**: Tracks `name`, `email`, `phone`, `startDateTime`, `guests`, `parkId`, and `parkName`.
    - **Auto-population**:
        - `parkId`/`parkName` from URL params (e.g., `?parkId=123&parkName=Yellowstone`).
        - `email` from authenticated users (via Firebase `onAuthStateChanged`).
2. **Validation & Submission**
    - **Prevents past dates**: Checks if `startDateTime` is valid.
    - **Firebase Storage**:
        - Saves under `/reservations/{dateKey}` (dateKey = YYYY-MM-DD).
        - Updates park capacity via `updateCurrentCapacity(parkName, guests)`.
    - **Redirect**: To `/my-reservations?email={email}` on success.
3. **UI Elements**
    - Required fields: `name`, `email`, `datetime-local`, `guests`.
    - Read-only park name display.
    - Submit/Cancel buttons.

Error Handling

- Alerts for:
    - Past reservation dates.
    - Firebase write failures.
- Console logs for debugging.

Dependencies

- **Firebase**: `database`, `auth`.
- **Routing**: `useLocation` for URL params.
- **Custom Hook**: `updateCurrentCapacity` (updates park capacity in DB).

ParkStatusReportForm Component Summary

A React form for submitting park status reports (open/closed) to Firebase with automatic data population.

Core Features

1. **Form Data Management**
    - **State**: Tracks `parkName`, `name`, `email`, `startDateTime`, `openOrClosed` (boolean), and `parkId`.
    - **Auto-population**:
        - `parkId`/`parkName` from URL params (e.g., `?parkId=123&parkName=Yellowstone`).
        - `email` from authenticated users (via Firebase `onAuthStateChanged`).
        - `startDateTime` defaults to current time.
2. **Submission Logic**
    - **Validation**: Rejects past dates.
    - **Firebase Storage**:
        - Saves under `/reports/{dateKey}` (dateKey = YYYY-MM-DD).
        - Updates park status via `updateParkStatus(parkName, openOrClosed)`.
    - **Redirect**: To `/map` on success.
3. **UI Elements**
    - Read-only park name display.
    - Checkbox for `openOrClosed` status (default: `true`/open).
    - Submit/Cancel buttons.

Key Functions

Function	Purpose
`handleInputChange`	Updates form state on input changes.
`handleSubmit`	Validates data, saves to Firebase, updates park status.