### Detailed Technical Design for NBA Player Management System

#### ****Goal****:

To create a responsive, interactive NBA player management system using React (with TypeScript), Material-UI, and Material-React-Table for the frontend, along with a backend API (in TypeScript/Node.js) that manages data fetching and pagination.

### ****Architecture Overview****

1. **Frontend (React + TypeScript)**
   * **Component Breakdown**: Splits the app into functional components.
   * **State Management**: Managed via React hooks (useState, useEffect).
   * **API Interaction**: Frontend interacts with the backend to fetch players list / stats and handles pagination, search, and error states.
2. **Backend (Node.js + TypeScript)**
   * **API Layer**: Exposes endpoints to fetch NBA player list and stats and potentially implement caching or rate limiting.
   * **Integration with External API**: Uses an external NBA API (e.g., balldontlie.io) to fetch real-time data.

### ****Frontend Design****

#### ****1. Component Breakdown****

##### ****App.tsx****

* **Responsibilities**:
  + Main entry point of the app.
  + Manages global state like the list of players, favorite players and search queries.
  + Controls API calls and handles UI updates (loading, errors, etc.).
  + Provides a responsive layout for mobile and desktop devices using Material-UI’s grid and CSS modules.
* **State**:
  + players: List of NBA players fetched from the backend.
  + favorites: List of favorite players added by the user.
  + favoritePlayer: Current added favorite player.
  + searchQuery: Current search term for filtering players.
  + loading: Loading state while fetching data from the API.
  + error: Stores error messages when API requests fail.
* **UI Components**:
  + **PlayerList MaterialReactTable** (Left column)
    - Shows the list of players with actions (Add to Favorites).
    - Handles pagination (10 items per page)
    - Implements search filtering.
  + **FavoriteList MaterialReactTable** (Right column)
    - Displays the list of favorite players added by the user with actions to remove them.
    - Handles pagination (10 items per page)

#### ****2. State Management Architecture****

* **Global State**:
  + Managed by the App component for now.
  + If the app scales, a state management solution like **Redux** or **Context API** could be introduced to handle the state globally across multiple components.
* **State Management Hooks**:
  + useState:
    - Used to manage the state of players, favorites,, search query, and error handling.
  + useEffect:
    - Used to handle side effects like fetching data when the page or search query changes or when user adds a favorite player.
* **Future Enhancements**:
  + If the app grows in complexity, state management can be centralized using Redux or React Context. We can also split the UI into 2 components: **PlayerList** and **FavoriteList**. These both components can refer to the players redux state.

#### ****3. API Interaction (Frontend)****

* **API Call Structure**:
  + The frontend interacts with the backend using Axios or Fetch to request NBA player data .

// Example API call in playerService.ts

export const fetchPlayers = async (searchQuery: string) => {

const response = await axios.get(API\_URL, {

params: {

searchQuery,

}

});

return response.data;

};

* **Error Handling**:
  + The frontend catches errors from the API calls and stores them in the error state, which is displayed using a Material-UI Alert component.
* **API Call Flow**:
  + **Trigger**: Search query change .
  + **API Call**: Frontend sends a request to /api/players with query parameter for search.
  + **Response Handling**:
    - On success: The player list is updated in the state.
    - On failure: The error state is updated, and an error message is displayed to the user.
* **API Call Structure**:
  + The frontend interacts with the backend using Axios or Fetch to request NBA stat data.
* // Example API call in statService.ts
* export const fetchStats = async (id: number) => {
* const response = await axios.get(API\_URL, {
* params: {
* id,
* }
* });
* return response.data;
* **Error Handling**:
  + The frontend catches errors from the API calls and stores them in the error state, which is displayed using a Material-UI Alert component.
* **API Call Flow**:
  + **Trigger**: new player added as favorite.
  + **API Call**: Frontend sends a request to /api/stats with query playerId.
  + **Response Handling**:
    - On success: The favorites list is updated with the stats of the new player.
    - On failure: The error state is updated, and an error message is displayed to the user.

### ****Backend Design****

#### ****1. API Endpoints****

##### ****GET /api/players****

* **Purpose**: Fetch NBA players from the external API.
* **Query Parameters**:
  + search: Search query for filtering players by name.
* **Flow**:
  + Makes a request to the external NBA API (e.g., balldontlie.io) with these parameters.
  + Returns player data in JSON format to the frontend.

##### ****GET /api/stats****

* **Purpose**: Fetch player state from the external API.
* **Query Parameters**:
  + id: id of the player .
* **Flow**:
  + Makes a request to the external NBA API (e.g., balldontlie.io) with these parameters.
  + Returns player stats in JSON format to the frontend. Actually we return average stats over up to 100 games for season 2023 (the only available for free tier).
  + We also cache the stats player into a table to avoid to call API again for the same player.

#### ****2. API Interaction (Backend)****

* **External API**: The backend fetches data from a third-party API (e.g., balldontlie.io), handling stats and search functionality.
* **Error Handling**:
  + If the external API fails or responds with an error, the backend will return a 500 status code with a descriptive error message.
* **Rate Limiting & Caching** (Future Consideration):
  + If necessary, caching (e.g., Redis) can be implemented to minimize API calls to the external service and improve response times.

### ****Backend System Components****

#### ****1. Express Router****

* **Role**: Serves the /api/players endpoint, accepts search parameters, and forwards the request to the controller.
* **Role**: Serves the /api/stats endpoint, accepts search parameters, and forwards the request to the controller.

### ****Integration Overview****

1. **Frontend**:
   * Sends requests to /api/players with search parameter.
   * Sends requests to /api/stats with id parameter whenever we add a new favorite player.
   * Displays player data in a paginated table and allows for adding/removing favorites.
   * Displays error messages when API calls fail.
2. **Backend**:
   * Interacts with the external NBA API to fetch player data and stats
   * Handles search, and potential errors (like rate limits).
   * Serves data to the frontend, ensuring a smooth user experience.

### ****Conclusion****

This design breaks down the application into modular components, making it easier to scale and maintain. The frontend uses React's component-based architecture with state management and error handling, while the backend exposes a clean API for the frontend to interact with, making the entire application modular, maintainable, and scalable.