**NBA Players UI Technical Design**

**Component breakdown:**

The client Application is made up of 3 components. The NBA Players lists which displays  
two lists. The first displays NBA players in a paginated list format (default 25) and allows  
the user to limit the amount of players shown on each page. The second list displays the   
statistics from the relevant players on the first list that are displayed.

The next component is the filter search which reduces the players list according to the characters entered by the user. The values entered are checked against first or last name.

The final component encapsulates the other 2 components thus allowing for a scalable  
form if more features will be included in the future.

Wrapper Component

Filter Component

Player Lists Component

**State Management Architecture:**

For Every call to the API, the data returned is placed in the store. For the first call to  
the API, the data is added to the state, and every additional returned data from  
the API is appended to the existing state. Each time the user clicks on the “next”  
or “previous” buttons the function checks whether the relevant page exists in the   
store. If it exists, the data is retured from the state. If the user changes the number  
of players to display the state is reset.

The state service has 3 main functions:

1. **getPlayers** – Gets a new data list from the API and stores it in the state
2. **getPlayersFromState** – Gets an existing data list from the state
3. **resetPlayersState** – resets the existing state

**Web API:**

The server exposes one main function called “GetPlayers” which for this task utilizes   
2 parameter, “cursor” and “per\_page”. The function calls an existing API to receive  
the players data and for each player calls another API to receive the players stats.  
The function returns an object containing all of the relevant data in JSON format.

**Overview:**

The stack I used for this task includes Angular 16(typescript) for client and C# for server side  
Web API. I used .css to implement my design and in a manner which supports  
responsivity. I structured my code in an optimal format to allow for reusability and  
robust performance. For example, I created a service to fetch data from the server  
and then used a technique called “dependency injection” to call it. To maximize  
memorization and performance, I used the NGXS library for state management.