**Question 1: How did you manage to fetch the list and what tool did you use?**

To retrieve the Pokémon list, I employed **Redux Toolkit Query (RTK Query)**, which is included in the **@reduxjs/toolkit** package. This tool facilitates API interactions by offering a declarative and efficient method for managing data retrieval and caching.

Here is a brief overview of the process:

1. **API Slice Configuration:** I established an API slice using createAPI from RTK Query. This slice specifies the base URL (https://pokeapi.co/api/v2/) and an endpoint for obtaining the Pokémon list (pokemon?limit=151).
2. **Hook Generation:** RTK Query automatically created a custom hook for data retrieval, specifically useGetPokemonQuery. This hook oversees the complete lifecycle of the request, encompassing loading states, error management, and caching.
3. **Integration:** I incorporated the hook into the App component to fetch the Pokémon list. The data is presented dynamically through a grid-based UI component, which is styled using Tailwind CSS.

By utilizing RTK Query, I ensured a streamlined process, benefiting from built-in caching and re-fetching capabilities for optimal performance.

**Question 2: What steps would you take to future improve this?**

To enhance and future-proof this implementation, I would concentrate on the following areas for improvement:

* 1. Error Handling and Retry Logic
* 2. State Normalization
* 3. Dynamic Caching Strategies
* 4. Lazy Loading
* 5. Improved UI/UX
* 6. Comprehensive Testing and Optimization

#### **Question 4: What makes createSlice in Redux Toolkit different than a reducer in Redux?**

1. **Declarative Syntax:** createSlice merges the functionalities of reducers and action creators into a single process, which significantly minimizes boilerplate code in comparison to conventional reducers.
2. **Immer Integration:** createSlice leverages Immer internally, enabling developers to write "mutating" code (like state.list = [...]) while maintaining immutability.
3. **Namespace Isolation:** Each slice is inherently scoped, which helps to avoid collisions in action types (for instance, pokemon/setPokemonList).
4. **Streamlined Action Management:** You don’t need to manually define action types or creators, as createSlice generates them automatically based on reducer function names.
5. **Immer Integration:** createSlice utilizes Immer internally, allowing developers to write code that appears to mutate state (such as state.list = [...]) while still preserving immutability.
6. **Streamlined Action Management:** There is no need to manually create action types or creators, as createSlice automatically generates them based on the names of the reducer functions.

#### **Question 5: Describe the Benefits of Immutable Code**

1. **Prevents Unintentional Mutations**
2. **Aids in State Troubleshooting**
3. **Promotes Consistent State Behavior**
4. **Enhances Re-Rendering Efficiency**
5. **Accommodates Functional Programming Principles.**

**Question 7: Explain the use of useEffect hook in React?**

The useEffect hook enables the execution of side effects within function components. Side effects encompass activities such as data retrieval, managing subscriptions, and manually altering the DOM, among others. The useEffect function is invoked subsequent to the component's rendering, guaranteeing that the DOM is refreshed prior to the execution of the side effect. It can be contingent upon particular values, prompting React to re-execute the effect whenever those values are modified.

**Question 8: What is a High Order Component (HOC)?**

A Higher-Order Component (HOC) is defined as a function that accepts a component as an argument and produces a new component that possesses augmented capabilities. HOCs serve as a design pattern for the reutilization of component logic. They do not alter the original component; instead, they encapsulate it to deliver supplementary functionality.

**Question 9: What use cases would an HOC be useful?**

Higher-Order Components (HOCs) serve as an effective means of encapsulating shared logic and implementing it across various components. Typical applications include:

* Authorization
* Logging
* Code Reusability
* State Management

**Question 10: What does it indicate when a component is prefixed with use?**

A component that begins with the prefix "use" is generally recognized as a custom hook in React. Custom hooks are functions designed for reuse that encapsulate and handle the stateful logic of React. They adhere to the same principles as the built-in hooks, such as useState and useEffect. These hooks facilitate code reuse while maintaining the integrity of the component hierarchy.

**Question 11: What is a Generic Type in TypeScript?**

A Generic Type enables the development of reusable components or functions that can operate with multiple types while ensuring type safety is preserved.

Flexibility: Function with diverse types while upholding type restrictions.

Reusability: Create a single implementation capable of managing various types.

**Question 12: What's the difference between a controlled and uncontrolled input in React?**

A controlled input is governed by the state in React, with its value linked to a specific state variable. The state is updated by React in response to user interactions with the input field.

An uncontrolled input, on the other hand, is regulated by the Document Object Model (DOM), and React does not oversee its state. To retrieve its value, a reference (ref) is utilized.