Offline Functionality Implementation

- State Persistence with Zustand Persist Middleware

Zustand's `persist` middleware is used to save fetched data in localStorage. This ensures data like character lists and details are preserved across sessions, reloads, and remain accessible offline.

- Local Caching with CharacterInfoRecords

`CharacterInfoRecords` uses a `Record<number, Character>` to cache character details by their IDs. This prevents redundant API calls and supports offline detail access.

- Early Return Optimization for Cached Data

Before making a network request for a character, the code checks if it already exists in the cache:
`if (characterDetails.data[id]?.id) return;`
This avoids unnecessary network calls and enables offline usage.

- Graceful Error Handling with errorMsg

Both `characters` and `characterDetails` include an `errorMsg` property to track errors during data fetches, helping to display fallback UI or retry prompts when offline.

- Separate Loading States for List and Detail

`isLoading` and `isFetchingNewCharacter` are independent loading states that provide accurate feedback for different operations, distinguishing between network operations and cached data usage.

- Encapsulation of Fetch Logic

Data-fetching is encapsulated in `fetchCharacters` and `fetchCharacterDetail` functions. This improves maintainability, testability, and supports both online and offline scenarios.

- Key Implementation Steps

- Imported dependencies: zustand, persist, API utility, and types.
- Defined interfaces: `CharactersState`, `CharacterInfoRecords`, and `CharacterStore` for structured state.
- Initialized Zustand store using `create` and wrapped with `persist`.
- Configured default state for characters and their detail records.
- Implemented `fetchCharacters`: handles API call for list, manages loading and error states.
- Implemented `fetchCharacterDetail`: checks cache before fetch, updates store, handles errors.
- Named persisted store as `character-store` for consistent key in localStorage.