A project implemented in React Native and oriented on the iOS platform will have typical ways of managing this storage. Among the usual strategies, the following are the most common ones:

1. **AsyncStorage**: AsyncStorage is a simple, asynchronous, unencrypted key-value storage system local to the app. It is used for small data requirements.
    - Pros:
        i. Provides a simple API to store small amounts of data.
        ii. This is asynchronous, it does not block the UI thread during data access.
        iii. Storage persisted through app launches.
    - Cons:
        i. Not suitable for large data sets since it might be slow and inefficient.
        ii. Does not have data encryption or security features put in place.
        iii. Clearing of data, when detected low storage conditions by the OS.

2. **Secure Store**: Secure Store allows you to encrypt and safely store key-value pairs. Normally, it is the way for sensitive data like passwords or personal information.
    - Pros
        i. Data is encrypted using robust cryptography techniques for security.
        ii. The API is easy and smooth to use, like AsyncStorage.
        iii. It gives more data security control than AsyncStorage.
    - Cons
        i. Performance overhead in the encryption and decryption process.
        ii. Data sizes would be limited in a similar way to AsyncStorage.
        iii. Complicated to set up and manage, unlike pure AsyncStorage.

3. **Realm**: Realms is an application database that is mobile-designed and poses a powerful yet easy alternative to SQLite. This helps to query complex relationships and support the objects of data.
    - Pros
        i. Large data set support with high performance.
        ii. It allows the creation of complex queries and relationships between the data objects.
        iii. Database that automatically updates UI components in real-time.
    - Cons
        i. The app size would increase with a more extensive library size.
        ii. Learning its database architecture and query language.
        iii. Can be overkill for simple storage needs.

4. **SQLite**: SQLite is an embeddable Relational Database Management System interfaced through various plugins to its interface in React Native.
    - Pros:
        i. A powerful and flexible SQL database system that is very well suited for the complex needs of data storage.
        ii. Very widely used, with a lot of community resources.

        iii. Can handle transactions with strong data-handling capabilities.
    - Cons:
        i. It is filled with boilerplate code and setup compared to the more straightforward key-value stores.
        ii. Direct operation with SQL can prone a user to errors.
        iii. Not as fast as some nonrelational databases for some operations.

5. **Redux Persist**: The library leverages Redux Persist to persist your redux store in a React Native application.
    - Pros:
        i. Integrates very well with Redux, and thus becomes the best friend to any project that is already in a relationship with Redux in state management.
        ii. This would be highly configurable using different storage backends and configurations.
        iii. Manages and persists application state across sessions effectively.
    - Cons:
        i. It is tightly coupled to Redux. Not valid if not using Redux. This may make state management challenging if not implemented properly.
        ii. Carefully needs handling to avoid bogging down performance issues during the app's startup.

My application manages storage using AsyncStorage from the **`@react-native-async-storage/async-storage`** package. This is preferred to others in that it is simple and efficient for persisting small amounts of data, for example, media files, across app sessions. AsyncStorage aligns with the needs of the media gallery app ideally since it is very convenient to use and, most importantly, it can retain the state across app launches, making it perfect for storing non-sensitive, straightforward data such as images and videos.

References:

- AsyncStorage. (n.d.). React Native documentation. Retrieved April 20, 2024, from https://reactnative.dev/docs/asyncstorage
- Expo Developers. (n.d.). SecureStore documentation. Retrieved April 20, 2024, from https://docs.expo.dev/versions/latest/sdk/securestore/
- MongoDB. (n.d.). Realm Database. Retrieved April 20, 2024, from https://www.mongodb.com/realm/mobile/database
- Andpor. (n.d.). React-native-sqlite-storage. GitHub repository. Retrieved April 20, 2024, from https://github.com/andpor/react-native-sqlite-storage
- rt2zz. (n.d.). Redux-persist. GitHub repository. Retrieved April 20, 2024, from https://github.com/rt2zz/redux-persist