**BÀI THU HOẠCH**

**Tìm hiểu về AsyncStorage và Context trong React Native**

**1. Tìm hiểu về AsyncStorage**

**Khái niệm và mục đích sử dụng**

* **AsyncStorage** là một API được cung cấp bởi thư viện @react-native-async-storage/async-storage trong React Native.
* Dùng để **lưu trữ dữ liệu cục bộ dạng key-value** trên thiết bị di động một cách **bất đồng bộ (async)** và **bền vững (persist)**, tương tự như localStorage trên web.
* **Mục đích sử dụng:** Lưu dữ liệu người dùng, token đăng nhập, thiết lập ứng dụng, giỏ hàng tạm thời...

**Cách hoạt động**

* AsyncStorage lưu trữ dữ liệu dưới dạng **key - value**.
* Hỗ trợ các phương thức như: setItem, getItem, removeItem, mergeItem, clear.
* Dữ liệu được lưu trữ vào hệ thống file của thiết bị hoặc bộ nhớ riêng biệt tùy nền tảng (Android/iOS).

**Đặc điểm**

* Bất đồng bộ, hoạt động dưới dạng Promise.
* Chỉ lưu được dữ liệu dạng **string** (cần serialize/deserialze khi lưu object hoặc array).
* Dữ liệu **lưu vĩnh viễn** đến khi bị xóa.

**Ưu điểm**

Lưu trữ dữ liệu lâu dài (persistent).  
Đơn giản, dễ dùng.  
Hỗ trợ bất đồng bộ giúp không chặn UI thread.  
Phù hợp lưu các token, dữ liệu không thường xuyên thay đổi.

**Nhược điểm**

Chỉ hỗ trợ kiểu dữ liệu string.  
Không phù hợp lưu trữ dữ liệu phức tạp hoặc dung lượng lớn.  
Không có cơ chế tự động cập nhật khi dữ liệu thay đổi.

**2. Tìm hiểu về Context (React Context API)**

**Context là gì?**

* Context API là một tính năng của React giúp **truyền dữ liệu giữa các component** mà **không cần thông qua props**.
* Giải quyết bài toán **"prop drilling"** khi dữ liệu cần truyền qua nhiều tầng component.

**Cách hoạt động**

* Gồm 3 phần chính:
  + **createContext()**: Tạo context.
  + **Provider**: Bao bọc component và cung cấp dữ liệu.
  + **Consumer / useContext()**: Component con sử dụng dữ liệu từ Provider.
* Khi giá trị context thay đổi, tất cả component sử dụng context đó **đều re-render**.

**Ưu điểm**

| **Tiêu chí** | **AsyncStorage** | **Context API** |
| --- | --- | --- |
| **Khái niệm** | Bộ nhớ cục bộ trên thiết bị | Cơ chế truyền dữ liệu trong component tree |
| **Lưu trữ dữ liệu** | Lưu vĩnh viễn trên thiết bị | Không lưu trữ, chỉ truyền dữ liệu tạm thời |
| **Bản chất** | Lưu dữ liệu (key-value) | Chia sẻ state giữa các component |
| **Dữ liệu bị mất khi reload?** | Không | Có |
| **Bất đồng bộ** | Có | Không |
| **Phạm vi sử dụng** | Lưu token, cấu hình app, data ít thay đổi | Lưu user info, theme, language... |
| **Tự động cập nhật UI** | Không | Có |
| **Dung lượng lưu trữ** | Có giới hạn (~6MB Android) | Không giới hạn (vì ở trong RAM) |

Truyền dữ liệu dễ dàng giữa các component.  
Giảm tình trạng prop drilling.  
Phù hợp cho các dữ liệu **thay đổi thường xuyên** như theme, language, user info...

**Nhược điểm**

Gây **re-render** không cần thiết nếu không tối ưu.  
Không lưu dữ liệu được khi tắt app hoặc reload trang.  
Không thay thế được state management phức tạp (Redux, Zustand...).

**3. So sánh AsyncStorage và Context**

**4. Khi nào nên dùng cái nào?**

**Nên dùng AsyncStorage khi:**

* Cần lưu dữ liệu **bền vững**, không bị mất khi tắt app hoặc reload như:
  + Token đăng nhập
  + Giỏ hàng
  + Cài đặt người dùng (dark mode...)
  + Lịch sử tìm kiếm, dữ liệu offline

**Nên dùng Context khi:**

* Cần chia sẻ **dữ liệu tạm thời** giữa các component:
  + Thông tin người dùng sau khi đăng nhập
  + Trạng thái đăng nhập
  + Theme, ngôn ngữ, socket connection...

**5. Thực tế sử dụng**

| **Trường hợp thực tế** | **Dùng AsyncStorage** | **Dùng Context** |
| --- | --- | --- |
| Lưu token đăng nhập | ✅ | ❌ Vì reload app mất dữ liệu |
| Truyền thông tin user trong app | Có thể ✅ kết hợp Context để lấy dữ liệu | ✅ |
| Giỏ hàng | ✅ | ❌ |
| Theme (Dark/Light Mode) | Có thể lưu AsyncStorage nhưng | ✅ dùng Context để thay đổi ngay lập tức |
| Socket connection | ❌ | ✅ |
| Lưu cài đặt app | ✅ | ❌ |

**Kết luận**

* **AsyncStorage**: Phù hợp **lưu dữ liệu lâu dài** trên thiết bị, không bị mất khi tắt app hoặc reload.
* **Context**: Phù hợp cho việc **chia sẻ dữ liệu tạm thời trong vòng đời app**, giúp tránh prop drilling và re-render tự động.

✅ Kết hợp cả hai để tối ưu:

* Lưu token bằng AsyncStorage.
* Khi load app, lấy token từ AsyncStorage rồi lưu vào Context để dùng trong app.