**Mobile Computing – Homework 2**

**Choice of database**

When it comes to managing storage in a Flutter app, there are several approaches we can take depending on our specific requirements, such as the size and type of data we need to store, the platform we’re targeting (Android, iOS, web), and whether the data needs to be persisted locally or stored remotely. Here are some common approaches to storage management:

1. Local Storage using SharedPreferences or Local Database:

For small amounts of data such as user preferences or settings, we can use SharedPreferences on Android and NSUserDefaults on iOS.

For more structured data or larger datasets, consider using a local database like SQLite, Moor, or Hive. These databases allow us to store and query data efficiently.

1. File Storage: For storing images or other large files locally, we can use the path\_provider package to get the directory for storing files on the device's filesystem.
2. Cloud Storage: If we need to store large amounts of data or share data between devices, consider using cloud storage solutions like Firebase Cloud Storage, AWS S3, or Google Cloud Storage.

These solutions provide scalable storage infrastructure, access control, and often include features like file versioning and CDN integration.

1. Cached Network Images: If our app frequently displays images from remote URLs, consider using packages like cached\_network\_image to efficiently cache images locally.

Caching images can improve app performance and reduce bandwidth usage by loading images from the local cache instead of downloading them every time.

1. Encryption and Security: If our app deals with sensitive data, consider encrypting the stored data using libraries like flutter\_secure\_storage or implementing our encryption algorithms.

**Pros and cons of using different storage options in our Flutter code:**

1. Local Storage using SharedPreferences or Local Database:

- Pros:

- SharedPreferences:

- Simple key-value pair storage.

- Lightweight and easy to implement for storing small amounts of data such as user preferences, settings, etc.

- No need for additional packages.

- Local Database (e.g., SQLite, Moor, Hive):

- Suitable for storing structured data and larger datasets.

- Offers more complex querying and data manipulation capabilities compared to SharedPreferences.

- Provides better performance for large datasets and complex data structures.

- Cons:

- SharedPreferences:

- Limited to primitive data types only (e.g., bool, int, string), which may not be suitable for complex data structures.

- Not suitable for storing large amounts of data.

- Local Database:

- Requires more setup and configuration compared to SharedPreferences.

- Overhead of database management and maintenance.

- Might be overkill for simple data storage needs.

2. File Storage:

- Pros:

- Suitable for storing various types of data such as images, videos, documents, etc.

- Provides flexibility in organizing and accessing files.

- Works well for scenarios where offline access to data is required.

- Cons:

- Managing files can become complex, especially when dealing with large numbers of files or complex directory structures.

- File operations can be slower compared to database operations, especially on devices with limited storage or slower storage mediums.

3. Cloud Storage:

- Pros:

- Offers scalability and accessibility from anywhere with an internet connection.

- Provides data backup and redundancy, reducing the risk of data loss.

- Suitable for collaborative applications where multiple users need access to the same data.

- Cons:

- Requires an internet connection for access, making it unsuitable for offline scenarios.

- May incur costs based on usage, especially for large amounts of data or high levels of access.

- Data security and privacy concerns, especially for sensitive data.

4. Cached Network Images:

- Pros:

- Improves app performance by reducing network requests and latency.

- Provides offline access to previously fetched images, enhancing user experience.

- Helps conserve bandwidth and reduce server load.

- Cons:

- Requires careful management of cache size and expiration policies to avoid consuming excessive device storage.

- May lead to stale data if not properly updated or refreshed.

- Can introduce complexity, especially when dealing with cache invalidation and consistency.

In our code, we are primarily using SharedPreferences for storing a simple boolean flag for dark mode preference, and File Storage for downloading and storing images locally. These choices seem appropriate for our use case, as SharedPreferences offer a lightweight solution for storing user preferences. At the same time, File Storage is suitable for caching downloaded images locally for offline access and improved performance. However, depending on the specific requirements of our application, we might consider incorporating other storage options like local databases or cloud storage for handling more complex data or larger datasets.