CSS 545A - Mobile Computing

HW2- Basic Storage

Raviteja Tanikella

There are mainly 5 approaches to manage the storage in iOS apps. They are:

1. **User Defaults**

- UserDefaults is a key-value store for storing user preferences and lightweight data.
- It's part of the Foundation framework and very easy to use for simple data types like strings, numbers, dates, and arrays.
- Data saved in UserDefaults is automatically written to disk periodically, but you can manually synchronize data if immediate storage is necessary.

Use Case in ATSPro:

- Storing user preferences, such as whether to receive notifications about new job matches.
- Remembering the user's last-used settings, like default language or theme preferences.

Pros:

- Simple to implement and use for storing small amounts of data.
- Data is quickly accessible and syncs across the user's devices if iCloud is enabled.

Cons:

- Not suitable for storing large amounts of data or sensitive information.
- Lack of structure, making it inefficient for handling complex data that goes beyond simple key-value pairs.

2. Core Data

- Core Data is not just a database but a comprehensive data management framework that provides object graph management.
- It allows for the definition of data models via an editor in Xcode where entities (like tables in databases) and their relationships can be visually mapped.
- Core Data handles all the interaction with the underlying SQLite database, if used as the store type, and can also manage data in memory or other custom store types.
- It provides features like data validation, lazy loading of data, and integration with SwiftUI and UIKit for automatic UI updates when data changes.

Use Case in ATSPro:

- Managing detailed user profiles, including their education, work history, skills, and applied jobs.
- Storing detailed feedback on resumes, linking them to specific job applications and the modifications suggested for each.

Pros:

- Robust data model configuration and versioning.
- Integrated with iOS, providing performance optimizations and scalability.
- Supports complex queries and relationships which is ideal for handling structured data like user profiles and resumes.

Cons:

- Complex setup and steep learning curve.
- Overhead for smaller datasets can be unnecessary.

3. Keychain

- Keychain is a secure storage container managed by the Secure Enclave on iOS devices.
- It's designed to store small pieces of sensitive data such as passwords, secure tokens, and encryption keys.
- Data in Keychain is encrypted and accessible only by the app that created it, or by other apps from the same vendor if configured to share access.

Use Case in ATSPro:

- Safely storing authentication tokens or credentials that a user might need to access web services.
- Protecting any sensitive personal information that must be retained across app installations.

Pros:

- Highly secure, suitable for storing sensitive data like passwords and tokens.
- Data persists even after the app is uninstalled, adding an extra layer of security.

Cons:

- Limited to small data sizes, not suitable for large data sets.
- Accessing keychain data can be slower compared to other methods.

4. SQLite:

- SQLite is a relational database management system contained in a C library that iOS supports natively.
- It is file-based, making it a great choice for offline data storage, and supports SQL syntax for data manipulation and querying.
- Unlike Core Data, you need to manage your SQL queries, database schema, indices, and handle migrations manually.

Use Case in ATSPro:

- Handling extensive offline data needs, such as a cache of job postings or industry-specific terminology that could be used for enhancing resume keywords.
- Supporting complex queries that integrate various data sources like user inputs and job market trends.

Pros:

- Great for larger data sets that do not fit into UserDefaults or require more structure than Core Data.
- Direct SQL access allows for complex queries and fine control over data.

Cons:

- Requires manual management of SQL queries, schema design, and data migration.
- Lack of built-in data consistency and concurrency mechanisms compared to Core Data.

5. CloudKit

- CloudKit is Apple's cloud backend service and database solution that provides a seamless way to sync user data across multiple iOS devices and store data in iCloud.
- It supports public and private databases; public data can be shared among all app users, while private data is specific to each iCloud account.
- CloudKit handles all the heavy lifting of data transfer, notifications, and synchronization across devices, making it highly scalable and efficient.

Use Case in ATSPro:

- Syncing user data like resumes and application statuses across multiple devices, ensuring that changes made on one device are reflected on all others.
- Enabling collaborative features where feedback or edits to resumes could be shared between users or career advisors securely.

Pros:

- Seamless integration with iCloud, providing automatic data sync across devices.

- Scalable and managed by Apple, reducing the need for server-side management.

Cons:

- Requires users to have an iCloud account and be signed in.
- Limited to the data storage provided by the user's iCloud account.

Considerations for ATSPro

Data Security: Since I'm handling sensitive user data (resumes), secure storage like Keychain for credentials and encrypted SQLite or Core Data for resume data is critical.

Data Complexity: Core Data is well-suited for handling complex data models involved in ATSPro.

User Data Sync: If I try to sync data across devices, integrating CloudKit could enhance user experience by keeping data consistent across all user devices.

Performance and Scalability: Core Data and SQLite both provide good performance, but Core Data offers additional iOS optimizations.