Tiba alramadan

Cs 499

Reflection on Weight Tracker Project Enhancements

Over the course of this project, I worked extensively on improving the database management, user authentication, and user interface of the Weight Tracker app. The main focus was on making the system more robust, secure, and maintainable, while also enhancing user experience. Below is a detailed reflection of the work completed:

1. Database Enhancements

Initially, the `UserDB` and `WeightDB` classes were functional but lacked several modern features and security measures. We implemented the following enhancements:

- **Singleton Pattern:**
  Both `UserDB` and `WeightDB` were converted into singletons to prevent multiple database instances, ensuring consistent access and reducing the risk of memory leaks.
- **Password Security:**
  User passwords were originally stored in plain text. We updated the system to use **SHA-256 hashing** for passwords. This ensures that sensitive user data is stored securely and improves overall security compliance.
- **Goal Management:**
  Added goal tracking functionality in `WeightDB` with methods to add, update, and retrieve goals per user. This allows the app to notify users when they meet their weight goals.
- **User Validation:**
  Replaced the old `checkUserPassword` method with `validateUser`, which returns the `userID` of a valid user. This streamlined the login process and simplified integration with `UserModel`.
- **Database Normalization:**
  Users, goals, and records were separated into three tables with proper foreign key relationships, improving data integrity and making future features easier to implement.

2. User Model Improvements

- The `UserModel` class was refined as a singleton to store information about the currently logged-in user, including their username, goal, SMS notification preferences, and user ID.
- This allows easy and consistent access to user information across different activities without redundant database calls.

3. GoalManager Implementation

- Introduced a dedicated `GoalManager` class to handle all goal-related operations.
- This separates business logic from database code, making the system easier to maintain and scale.
- Users can now update their goals, and the changes are directly reflected in the database.

4. WeightDB Enhancements

- Methods like `getAllWeights`, `addEntry`, `removeEntry`, and `updateEntry` were refined to work efficiently with `UserModel`.
- Added proper error handling and closed all database cursors to prevent memory leaks.
- Weight records are now retrieved in descending date order for better display in the UI.

5. Edit View and UI Enhancements

- The `edit_view` activity was improved to allow users to **select multiple records** and either delete or edit them efficiently.
- Used dynamic `TableLayout` rows with checkboxes and listeners for selecting rows.
- Edit functionality now prompts users for a new weight and updates the database in real-time.
- Implemented a refresh mechanism after editing or deleting records to ensure the UI reflects the current database state.

6. Weight Entry Form

- Updated the `weight_entry` activity to integrate with the updated `WeightDB` and `UserModel`.
- Added a date picker for user-friendly date input.
- Added checks for empty input fields to prevent crashes and errors.
- Integrated SMS notifications for when users reach their weight goals.

7. Login and Authentication Improvements

- Updated the login mechanism to use `validateUser` from `UserDB`.
- This ensures that authentication is secure and reduces reliance on direct password comparisons.
- The login flow is now consistent with the singleton `UserModel` approach, keeping the current user session intact across activities.

8. Overall System Reflection

- The project evolved from a basic weight tracker to a **secure, modular, and user-friendly application**.
- Code readability and maintainability improved significantly due to separation of concerns (`GoalManager`, `WeightDB`, `UserDB`) and adherence to best practices (singleton patterns, password hashing, cursor management).
- Future enhancements, such as analytics, charts, or push notifications, can be integrated seamlessly due to the improved architecture.

Key Takeaways:

1. Security is critical in apps dealing with personal data — password hashing and proper database structure are essential.
2. Singleton patterns simplify state management in Android apps, especially for databases and user sessions.
3. Separating business logic from database operations improves maintainability and scalability.
4. User experience is improved by dynamic UI elements and real-time updates after data changes.

5. Debugging and updating legacy code can be complex, but incremental improvements ensure stability and robustness.