**Food Journals App Report**

This report details the bugs found and fixes implemented during the development of the app. It also presents the app's functionality and appearance.

***Error Applying Method to Wrong Element***

* **Problem:** The initial code attempts to call execAsync on the transaction object tx, which is incorrect in expo-sqlite. The execAsync method is only available on the database object db.
* **Cause:** The expo-sqlite library's API dictates that execAsync is a method of the database object, not the transaction object. Transaction objects typically use a different method, like executeSql (as shown in the previous response when discussing reintroducing transactions), to execute SQL commands within a transaction.
* **Solution:** The corrected code removes the incorrect usage of tx.execAsync and replaces it with db.execAsync. Also, the transaction object tx is not being used, so it can be removed from the parameters: async (tx) => becomes async () =>.

**Initial Code:**

**Изображение выглядит как текст, снимок экрана, Шрифт, документ

Контент, сгенерированный ИИ, может содержать ошибки.**

**Final Code:**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Key improvements and explanations:**

* **db.withTransactionAsync:** The code now correctly uses db.withTransactionAsync to ensure that the table creation operations are performed within a transaction. This is important for data integrity, as it ensures that both tables are created successfully, or neither is created.
* **db.execAsync:** Inside the transaction, db.execAsync is used to execute the CREATE TABLE statements. This is the correct way to execute raw SQL commands directly on the database object.
* **Simplified Arrow Function:** The arrow function passed to db.withTransactionAsync no longer declares the tx parameter because it is not being used. This makes the code cleaner and more readable.
* **Idempotent Table Creation:** The CREATE TABLE IF NOT EXISTS syntax ensures that the tables are only created if they don't already exist. This prevents errors if the code is run multiple times.
* **Foreign Key Constraint:** The FOREIGN KEY(userId) REFERENCES users(id) clause establishes a relationship between the journals table and the users table. This ensures that the userId column in the journals table always references a valid id in the users table, maintaining referential integrity.

**Why This Approach is Recommended for Table Creation**

Using db.withTransactionAsync in conjunction with db.execAsync for creating tables is a good practice for the following reasons:

* **Atomicity:** If one of the CREATE TABLE statements fails (e.g., due to a syntax error or a conflict with an existing table), the entire transaction will be rolled back, ensuring that the database remains in a consistent state.
* **Isolation:** The table creation operations are isolated from other database operations that might be happening concurrently. This prevents interference and ensures that the tables are created correctly.
* **Durability:** Once the transaction is committed, the table creation operations are guaranteed to be durable, meaning that they will survive even if the application crashes or the device is restarted.

**Important Considerations**

* **Error Handling:** While the code includes a transaction, it lacks explicit error handling within the transaction block. It's a good practice to add try...catch blocks inside the withTransactionAsync callback to handle potential errors during table creation. If an error occurs, you can use tx.rollback() to roll back the transaction and prevent any changes from being committed.
* **Asynchronous Operations:** The execAsync method is asynchronous, which means that it returns a promise. The code uses await to ensure that each CREATE TABLE statement is executed before the next one. This is important because the journals table has a foreign key dependency on the users table, so the users table must be created first.

**Summary**

The execAsync method is used incorrectly on a transaction object in the initial code. The solution replaces tx.execAsync with db.execAsync and removes the unused tx parameter. This approach, combined with db.withTransactionAsync, provides a reliable way to create database tables while ensuring atomicity, isolation, and durability. Adding error handling, as demonstrated in the example, further enhances the robustness of the code.

***Fixes executeSQL Function***

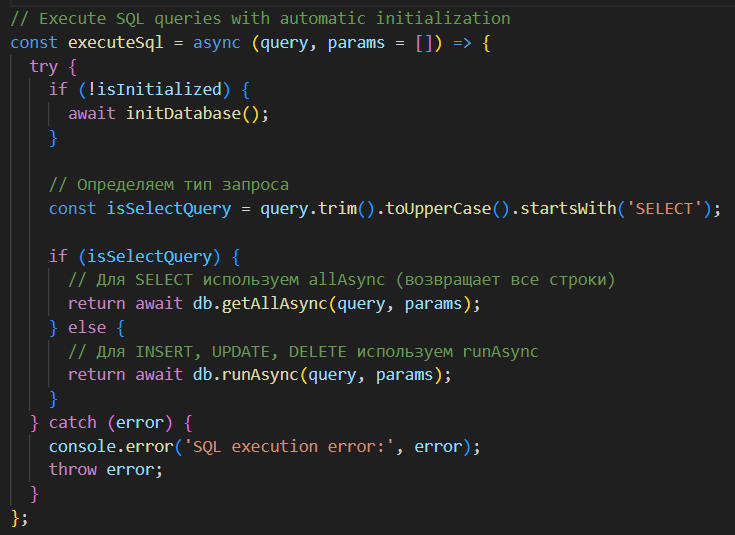
* **Problem:** The original executeSql function incorrectly attempts to use tx.execAsync within a transaction. execAsync is not a valid method on the transaction object. Additionally, wrapping every query in a transaction is unnecessary overhead for simple queries.
* **Cause:** The expo-sqlite library provides distinct methods for executing SQL commands. execAsync is designed to be used directly on the database object (db), while transaction objects typically use executeSql. The original code also lacked a mechanism to differentiate between SELECT queries (which return data) and INSERT, UPDATE, or DELETE queries (which modify data).
* **Solution:**The corrected code removes the unnecessary transaction wrapper and uses db.getAllAsync for SELECT queries and db.runAsync for INSERT, UPDATE, and DELETE queries. This approach aligns with how expo-sqlite is intended to be used for single-query operations.

**Initial Code:**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

**Final Code:**



**Key elements of the corrected code:**

* **Initialization Check:** Ensures the database is initialized before any operations are performed.
* **Query Type Detection:** Determines if the query is a SELECT statement by checking if the trimmed, uppercase version of the query string starts with "SELECT".
* **db.getAllAsync for SELECT:**Executes SELECT queries using db.getAllAsync, which retrieves all rows from the result set as an array of objects.
* **db.runAsync for Data Modification:** Executes INSERT, UPDATE, and DELETE queries using db.runAsync. This method is appropriate for queries that modify data but don't necessarily return rows.
* **Error Handling:** Includes a try...catch block to handle potential errors during database operations. Errors are logged to the console and then re-thrown, allowing the calling function to handle them.

**Advantages of the Corrected Approach**

* **Correctness:** Fixes the fundamental error of using tx.execAsync, aligning with the expo-sqlite API.
* **Efficiency:** Removes the overhead of unnecessary transactions for single-query operations.
* **Clarity:** Separates the handling of SELECT queries (data retrieval) from data modification queries, making the code more readable and maintainable.

**When to Consider Transactions**

While the corrected code is suitable for many scenarios, you should consider using transactions in the following cases:

* **Multiple Related Operations:** When you need to perform multiple SQL operations as a single atomic unit (i.e., all operations must succeed, or none should).
* **Data Consistency:** When data consistency is critical and you need to ensure that changes are applied in a reliable manner.
* **Complex Operations:** When performing complex operations that involve multiple tables or steps.

If you need to use transactions, you would use db.transactionAsync and tx.executeSql (as demonstrated in the previous answer), taking care to handle the result sets appropriately.

**Summary**

The corrected executeSql function addresses the original error by using db.getAllAsync for SELECT queries and db.runAsync for other queries. This approach is more efficient and aligns with the expo-sqlite API for single-query operations. Transactions should be used when atomicity and data consistency are paramount for multi-step operations.

***Removing Incorrect Formatting and Adding Error Handling***

* **Problem:** The code is incorrectly accessing the journal data using .rows.\_array, which is unnecessary and potentially incorrect since the data is already an array. Also, the error logging lacks detail, specifically the error message.
* **Cause:** The structure of the data returned from the database query might not match the expectation of the code. The .rows.\_array access pattern is likely based on an older version or incorrect understanding of the expo-sqlite API or a misunderstanding of how the data is structured after the query. Insufficient error logging makes debugging difficult.
* **Solution:** Remove the .rows.\_array part to directly use the array of journal data and add error.message to the console.log statement to provide more informative error messages.

***Changing the Working Items of the Library***

* **Problem:** The existing homeScreen.js code uses the older expo-camera API, which is incompatible with the current version of the library. Specifically, it relies on a Camera component and a different method for handling permissions.
* **Cause:** The expo-camera library has evolved, and the Camera component has been replaced with CameraView. The permission handling has also been updated to use the useCameraPermissions hook. The way refs are handled in React has also shifted towards using the useRef hook for managing mutable references.
* **Solution:** Update the code to use CameraView instead of Camera, implement permission handling with useCameraPermissions, and manage the camera's ref using useRef.

Finally, let's look at the structure and functionality of the application, presented below, as well as in the README file.

Изображение выглядит как текст, снимок экрана, документ, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

In addition, let's take a look at what the application actually looks like.

Изображение выглядит как текст, гаджет, Устройство связи, Мобильное устройство

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как Мобильный телефон, гаджет, Устройство связи, Электронное устройство

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, мультимедиа, Мобильный телефон, Устройство связи

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, мультимедиа, снимок экрана, Устройство связи

Контент, сгенерированный ИИ, может содержать ошибки.

In addition, I present a link to the repository: <https://github.com/KristinaAnd/Food-Journals.git>