Ministry of Science and Higher Education of the Russian Federation
"Ural Federal University named after the first President of Russia B.N. Yeltsin"
Institute of Radio electronics and Information Technologies – RTF

Ural Federal University
Institute of radioelectronics and information technologies

Food Journal

Report Project

Student : AYAD Seif Eddine

Group: RIM-140930

Date: 25/05/2025

Github Project Link : https://github.com/SifouAyad007/Journalfood

Contents

Chapter I – HomeScreen.js Analysis

This chapter details the findings from a review of the HomeScreen.js implementation, which revealed several architectural and functional inconsistencies between the incorrect and correct versions.

1.1 Database Schema Alignment

The older version of the SQL queries used incorrect table and field names, such as "journals" and "description," which caused query failures and data issues due to schema mismatches. This has been resolved by updating the queries to match the correct schema, using names like "journal," "imageUri," and "text" or "desc."

1.2 Camera Reference Handling

The Camera was being managed with useState instead of useRef, leading to unexpected behavior and reference errors in the Camera component.

- **Resolution**: Replaced with const cameraRef = useRef(null) and passed appropriately via ref.

1.3 Camera Readiness Check

There was no readiness check before capturing an image, which could lead to runtime errors or capturing empty images.

- **Resolution**: Introduced isCameraReady state and onCameraReady callback to ensure readiness before action.

1.4 Media Library Permissions

The `pickImage` function had redundant permission requests, causing a poor user experience and potential conflicts in permission handling..

- **Resolution**: Request and store permissions once in useEffect.

1.5 Loading State Management

The `isLoading` state was mismanaged, resulting in incorrect or missing visual loading indicators during operations.

- **Resolution**: Controlled state flow with accurate `true/false` transitions during data load lifecycle.

1.6 Data Integrity in SQL Operations

The DELETE and UPDATE statements lacked userId validation, posing a risk of modifying or deleting data belonging to other users.

- **Resolution**: Updated queries to include `userId` in `WHERE` clauses.

```
sql
CopyEdit
DELETE FROM journal WHERE id = ? AND userId = ?
```

1.7 Field Name Consistency

There was a misalignment between the code and database field names, causing query failures and potential data corruption.

- **Resolution**: Standardized field references to match schema (`imageUri`, `desc`/`text`).

1.8 Safe Image Picker Handling

The image picker result was accessed without checking if the user canceled, which could cause the application to crash.

- **Resolution**: Safely check `!result.canceled` before processing results.

1.9 Form Reset State Management

The form reset logic was incomplete, causing stale state values to persist after actions.

- **Resolution**: Enhanced `resetForm()` to reset all relevant states including `category`, `image`, `description`, and `editingId`.

Chapter II – Database Integration and Management

This chapter reviews the handling of SQLite database connections, setup, and transactions in `database_wrong.js` vs. `database.js`.

2.1 API Misuse: Async/Await with expo-sqlite

Asynchronous functions like `openDatabaseAsync` and `execAsync`, which are not supported by expo-sqlite, were used incorrectly, leading to runtime errors and failed database access.

- **Resolution**: Replaced with synchronous API:

```
javascript
CopyEdit
const db = openDatabase('myDB.db');
db.transaction(tx => {
    tx.executeSql('...');
});
```

2.2 Database Initialization

The async logic in the setup failed because of incorrect API usage, preventing proper initialization.

- **Resolution**: Converted setup to synchronous format using `db.transaction`.

2.3 SQL Execution Strategy

The async/await syntax was used incorrectly with `executeSql`, causing execution failures and unhandled promise rejections.

- **Resolution**: Implemented callback-based execution inside a wrapped `Promise` for controlled async behavior.

2.4 Initialization Dependency in Queries

Calls to initDatabase() were inconsistent before running SQL queries, resulting in an uninitialized database and query failures.

- **Resolution**: Check and initialize DB status before query execution reliably.

2.5 Error Handling in SQL Operations

Errors within async functions were not properly propagated, causing them to be swallowed or misdirected and making debugging difficult.

- **Resolution**: Moved error logic into `executeSql` callbacks with proper rejection.

2.6 PRAGMA Statement Usage

- **Observation**: Only correct version uses `tx.executeSql('PRAGMA journal_mode = WAL')` effectively.

- **Resolution**: Confirmed correct implementation for enabling Write-Ahead Logging mode.

Chapter III – Authentication Module Review

The following chapter outlines discrepancies and corrections in the authScreen_wrong.js and authScreen.js files.

3.1 Input Validation Logic

Conditional blocks in validateInputs were improperly closed, leading to unreliable validation and missed error handling.

- **Resolution**: Structured and closed all logic blocks clearly to ensure robust validation.

3.2 Query Result Interpretation

There was a mistaken assumption that the result returned a WebSQL-like structure, causing null reference or data access errors.

- **Resolution**: Adjusted logic to treat query result as a flat array:

```
javascript
CopyEdit
if (users.length > 0) {
    const userId = users[0].id;
}
```

3.3 Registration Logic Error Isolation

Using a single try-catch block for all database operations led to undifferentiated error handling and made it harder to identify specific issues.

- **Resolution**: Separated registration logic into nested try-catch blocks for specific error tracing.

3.4 UI Component Rendering

The JSX was improperly structured with dangling fragments, causing the component to fail to render.

- **Resolution**: Fully reconstructed JSX return with valid syntax and conditional rendering.

3.5 User Experience: Keyboard Management

Although imported, KeyboardAvoidingView was not implemented, causing the keyboard to overlap input fields on mobile devices.

- **Resolution**: Wrapped input components with `KeyboardAvoidingView` for improved UX.

3.6 Code Consistency and Style

Inconsistent formatting and syntax errors, like missing semicolons, reduced code readability and made maintenance more difficult.

- **Resolution**: Adopted consistent style guides, enforced proper indentation, semicolons, and naming conventions.

Conclusion & Recommendations

The audit revealed multiple critical issues, predominantly centered around improper API usage, inconsistent schema alignment, and flawed control flow in both UI and backend integration. These issues pose a significant risk to functionality, data integrity, and user experience.

Recommended Actions

We refactor the codebase to ensure proper API usage and follow architectural standards, centralizing schema definitions in a configuration file to prevent field name mismatches. Improving testing with unit and integration tests for SQL operations and UI forms strengthens reliability, while structured peer code reviews help catch issues early. Clear documentation of platform-specific permission handling supports consistent implementation across the project.