Mobile App Development Report - Food Journal

In this report, we will look at what kind of application we got and what bugs we encountered during our work. First, let's see how the app looks like externally and what features it has (Figure 1-3).

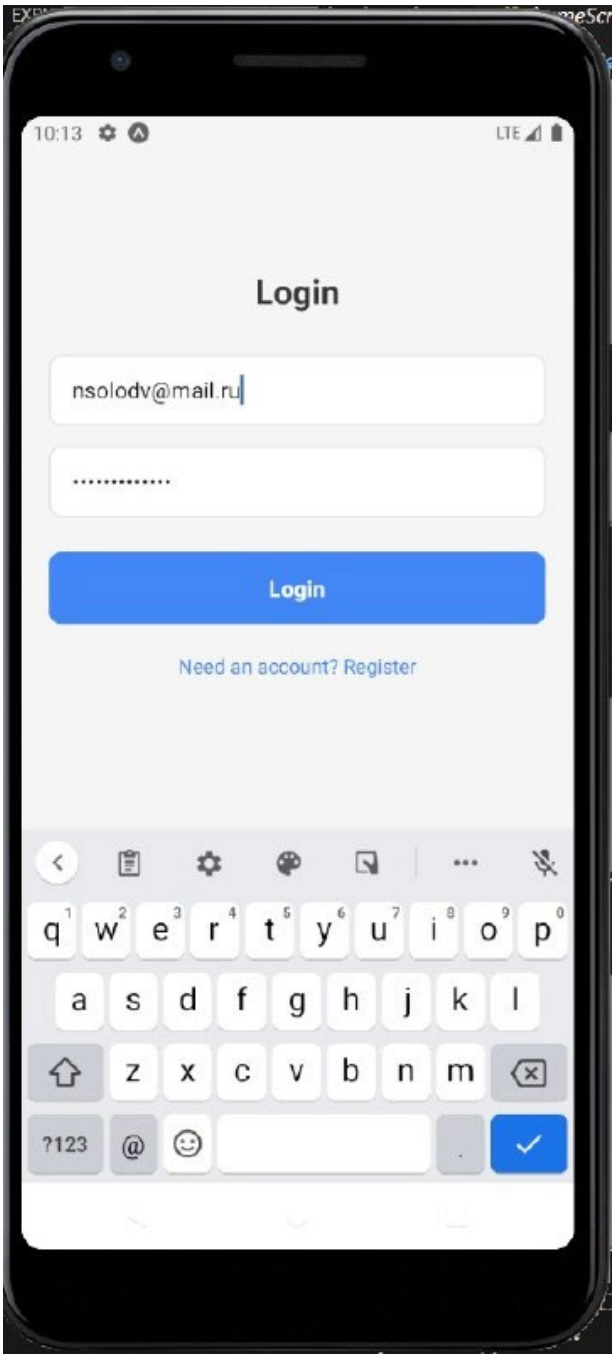

Figure 1 – Account Login

Figure 2 – Taking of the picture

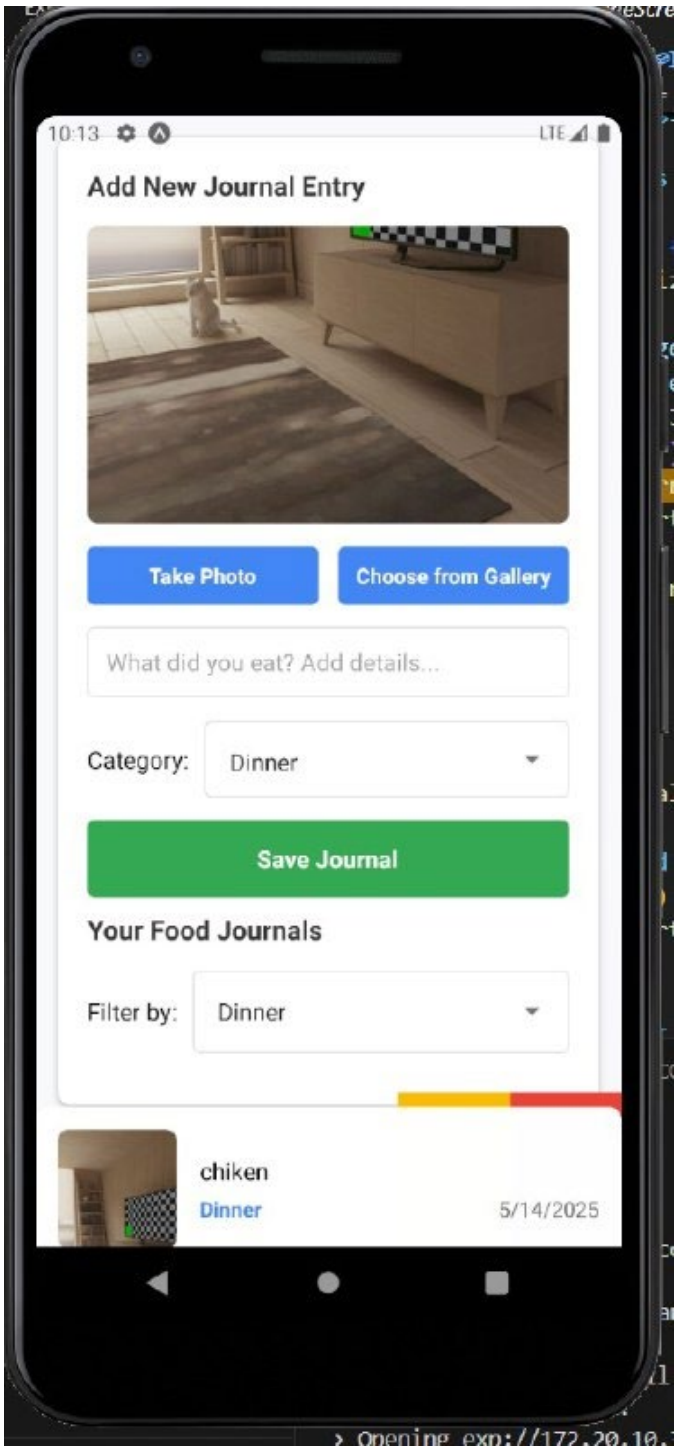

Figure 3 – Category selection and filtering

Next, we need to consider the mistakes encountered during the creation of the mobile application and how to solve them.

1. Error Applying Method to Wrong Element: the source code (figure 4) attempts to call the execAsync method for the transaction object tx, but this execAsync method can be used exclusively for the database object db => according to the expo-sqlite library API, the execAsync method is part of a

database object, not a transaction object; to execute SQL commands within a transaction, another method is usually used, for example executeSql => the corrected version of the code (figure 5) eliminates the incorrect use of the tx.execAsync method, which is replaced by db.execAsync, since the transaction object tx is not involved, it can be excluded from the parameters by converting async (tx) => to async () =>.

```
await db.withTransactionAsync(async (tx) => {
      await tx.execAsync(
        `CREATE TABLE IF NOT EXISTS users (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          email TEXT UNIQUE,
          password TEXT
        );`
      );

  await tx.execAsync(
    `CREATE TABLE IF NOT EXISTS journals (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    userId INTEGER,
    image TEXT,
    description TEXT,
    date TEXT,
    category TEXT,
    FOREIGN KEY(userId) REFERENCES users(id)
    );`
  );
}
```

Figure 4 - The source code

```
await db.withTransactionAsync(async () => {
  await db.execAsync(
    `CREATE TABLE IF NOT EXISTS users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      email TEXT UNIQUE,
      password TEXT
    );`
  );

  await db.execAsync(
    `CREATE TABLE IF NOT EXISTS journals (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      userId INTEGER,
      image TEXT,
      description TEXT,
      date TEXT,
      category TEXT,
      FOREIGN KEY(userId) REFERENCES users(id)
    );`
  );
});
```

Figure 5 – The corrected version of the code

2. Fixes executeSQL Function: the original executeSql function (figure 6) erroneously applies tx.execAsync within a transaction, but execAsync is not supported for transaction objects and encapsulating each query in a transaction creates an excessive load, especially for elementary operations; the original code also did not realize the distinction between SELECT queries (which return data) and INSERT, UPDATE, or DELETE queries (which modify data) => the expo-sqlite library offers a variety of methods for executing SQL commands; the execAsync method is intended for direct use with a database object (db), whereas transaction objects typically use the executeSql method => the corrected version of the code (figure 7) removes the redundant transactional shell and uses db.getAllAsync method to execute SELECT queries, while db.runAsync is used for INSERT, UPDATE and DELETE operations.

```
const executeSql = async (query, params = []) => {
  try {
    if (!isInitialized) {
      await initDatabase();
    }

    return await db.withTransactionAsync(async (tx) => {
      return await tx.execAsync(query, params);
    });
  } catch (error) {
    console.error('SQL execution error:', error);
    throw error;
  }
};
```

Figure 6 – The source code

```
// Execute SQL queries with automatic initialization
const executeSql = async (query, params = []) => {
  try {
    if (!isInitialized) {
      await initDatabase();
    }

    // Определяем тип запроса
    const isSelectQuery = query.trim().toUpperCase().startsWith('SELECT');

    if (isSelectQuery) {
      // Для SELECT используем allAsync (возвращает все строки)
      return await db.getAllAsync(query, params);
    } else {
      // Для INSERT, UPDATE, DELETE используем runAsync
      return await db.runAsync(query, params);
    }
  } catch (error) {
    console.error('SQL execution error:', error);
    throw error;
  }
};
```

Figure 7 – The corrected version of the code

3. Removing Incorrect Formatting and Adding Error Handling: ehe code contains incorrect access to the log data via .rows._array, which is redundant and may cause errors because the data is already represented as an array, and also the error logs lack important details such as the text of the error message => the use of the .rows._array construct is likely due to an outdated version or misinterpretation of the expo-sqlite API, or a misunderstanding of the organization of the data after the query is executed => should remove the .rows._array construct for direct access to the log data array, and include error.message in the console.log call, which will allow for more meaningful error messages.
4. Changing the Working Items of the Library: The current implementation in homeScreen.js relies on the deprecated expo-camera API, which is not supported by the current version of the library => the expo-camera library has undergone changes and the Camera component has been replaced with CameraView, and the permissions handling has been updated to use the useCameraPermissions hook, as has the way React handles references changed to use the useRef hook to manage changeable references => need to

update the code to use CameraView instead of Camera, implement permission handling with useCameraPermissions and manage the camera reference with useRef.

Журналы о еде

Журналы о еде — это мобильное приложение, созданное на React Native и Expo для отслеживания потребления пищи с помощью ведения журнала с изображениями и описаниями.

Функции:

- Регистрация и вход в систему с использованием локальной аутентификации SQLite.
- Добавление записей в журнал о еде с фотографией (с камеры или из галереи), описанием и категорией (завтрак, обед, ужин, закуски).
- Фильтрация записей в журнале по категориям.
- Редактирование и удаление записей.
- Локальное хранилище данных с использованием expo-sqlite.
- Совместимость с Expo Go (не требуется клиент разработки).

Функции:

- Аутентификация: регистрация/вход с использованием электронной почты и пароля.
- Камера/галерея: добавление изображений с камеры устройства или из галереи изображений.
- Ведение журнала: создание, обновление и удаление записей в журнале питания.
- Фильтр: фильтрация сообщений по категориям блюд.
- Автономная работа: работа в автономном режиме с использованием локального SQLite.

Установка:

1. Установите Node.js и npm, а также Expo CLI глобально (npm install -g expo-cli).
2. Клонируйте репозиторий:
3. Установите зависимости: `npm install`
4. Установите необходимые модули Expo: `npx expo install expo-sqlite expo-image-picker @react-native-picker/picker react-native-swipe-list-view`
5. Запустите проект: `npx expo start`
6. Отсканируйте QR-код в терминале с помощью Expo Go на вашем устройстве Android/iOS.

Figure 8 – README file

https://github.com/AnastasiaSolod/Food-Journals.git