Sprawozdanie Projektu: DailyFlow (Genesis Document)
Wersja: 2.0 (Kompletna)
Data: 03.08.2025

Cel dokumentu: Stanowi kompletną, wyczerpującą i aktualną bazę wiedzy o projekcie. Zastępuje wszystkie poprzednie wersje podsumowań i jest przeznaczony do użycia jako **jedyne, nadrzędne źródło informacji** w przyszłych interakcjach i dla przyszłych deweloperów.

---

1. Wizja, Misja i Cel Projektu

DailyFlow to kompleksowa, cross-platformowa aplikacja mobilna (iOS/Android) zaprojektowana jako zintegrowane centrum organizacji życia codziennego. Głównym celem jest zastąpienie potrzeby korzystania z wielu oddzielnych aplikacji poprzez połączenie zaawansowanego menedżera zadań, systemu zarządzania budżetem oraz narzędzi do współpracy w jednym, spójnym ekosystemie.

- **Grupa Docelowa**: Aplikacja jest skierowana głównie do par i partnerów mieszkających razem, którzy chcą w prosty i transparentny sposób zarządzać wspólnymi obowiązkami domowymi i finansami.

Kluczowe filary:

- **Zintegrowane Zarządzanie**: Użytkownik w jednym miejscu zarządza listą zadań i obowiązków oraz śledzi cele finansowe w ramach budżetów. Eliminuje to potrzebę przełączania się między aplikacją to-do a aplikacją do budżetowania.

- **Inteligentna Priorytetyzacja**: Aplikacja aktywnie pomaga użytkownikom skupić się na najważniejszych zadaniach. Jest to osiągane przez dynamiczny algorytm w `HomeScreen.tsx`, który oblicza priorytet zadania na podstawie jego terminu, "wieku" oraz w pełni konfigurowalnych przez użytkownika mnożników w `SettingsScreen.tsx`.

- **Współpraca w Parze**: Rdzeniem aplikacji jest możliwość połączenia kont z partnerem. Umożliwia to tworzenie wspólnych list zadań i budżetów, co promuje transparentność, ułatwia podział obowiązków i wspólne planowanie finansów w gospodarstwie domowym.

- **Motywacja i Personalizacja**: Elementy grywalizacji (punkty za wykonane zadania widoczne w `ProfileScreen.tsx`) oraz głęboka personalizacja (własne kategorie z kolorami, szablony często wykonywanych obowiązków, awatary z konta Google) mają na celu budowanie pozytywnych nawyków i zwiększenie zaangażowania użytkowników.

---

2. Stos Technologiczny i Środowisko Pracy

Framework i Język
- **Framework**: React Native z **Expo SDK 53**.
- **Język**: TypeScript (z włączoną opcją `"strict": true` w `tsconfig.json`).
- **Architektura**: Projekt jest skonfigurowany do użycia Nowej Architektury React Native (**Fabric**), zdefiniowanej przez flagę `newArchEnabled: true` w `app.json`.

Backend i Baza Danych
- **Dostawca**: **Firebase**
- **Usługi**:
  - `Firestore`: Jako główna baza danych NoSQL.
  - `Firebase Authentication`: Do zarządzania uwierzytelnianiem.

Kluczowe Biblioteki i Narzędzia
- **Nawigacja**: `React Navigation` (v7), w tym `@react-navigation/native-stack` i `@react-navigation/bottom-tabs`.
- **Natywne Moduły Firebase**: `@react-native-firebase/app` i `@react-native-firebase/auth`.
- **Logowanie Społecznościowe**: `@react-native-google-signin/google-signin`.
- **Animacje i Gesty**: `react-native-reanimated`, `react-native-gesture-handler`.
- **Pamięć Lokalna**: `@react-native-async-storage/async-storage` do buforowania zadań.
- **Komponenty UI**: `@expo/vector-icons`, `@react-native-community/datetimepicker`, `@react-native-community/slider`, `@react-native-picker/picker`.

Środowisko i Proces Deweloperski
- **System operacyjny**: Windows (Terminal: PowerShell).
- **IDE**: Android Studio (dla emulatora i narzędzi natywnych).
- **Testowanie**: Expo Dev Client na fizycznym urządzeniu z Androidem.

- **Środowisko Java**: JDK skonfigurowane z systemowymi zmiennymi środowiskowymi `JAVA_HOME` i `Path`.
- **Workflow Expo (Bare Workflow)**:
  1. Kod pisany jest w folderze `src`.
  2. Foldery natywne (`android`, `ios`) są generowane za pomocą komendy `npx expo prebuild`.
  3. Aplikacja jest budowana i instalowana na fizycznym urządzeniu jako niestandardowy klient deweloperski (`.apk`) za pomocą komendy `npx expo run:android`.
  4. Zmiany w kodzie JavaScript są odświeżane przez Metro, natomiast zmiany w kodzie natywnym lub instalacja nowych bibliotek z natywnymi modułami wymagają ponownego przebudowania aplikacji (ponownego uruchomienia `npx expo run:android`).

---

3. Aktualny Stan Aplikacji (Szczegółowy Opis Funkcjonalności)

Moduł Uwierzytelniania
- **System Wielu Dostawców**: Użytkownicy mogą logować się i rejestrować za pomocą:
  - **E-mail/Hasło**: Z walidacją w czasie rzeczywistym i polityką silnych haseł.
  - **Google**: Pełna integracja z Google Sign-In.
  - **Numer Telefonu**: Z weryfikacją przez kod SMS.
- **Identyfikacja Użytkownika**: Zunifikowane pole logowania (`identifier`) pozwala na użycie e-maila lub numeru telefonu. **Świadomie usunięto możliwość logowania przez nick** w celu uproszczenia logiki.
- **Odzyskiwanie Konta**: Wdrożono w pełni funkcjonalny mechanizm "Zapomniałem hasła" (`ForgotPasswordModal.tsx`), który wykorzystuje `sendPasswordResetEmail` Firebase.
- **Weryfikacja E-mail**: Nowi użytkownicy muszą zweryfikować swój e-mail, klikając w link aktywacyjny. Aplikacja blokuje logowanie niezweryfikowanym kontom i oferuje ponowne wysłanie linku.
- **Onboarding**: `AppNavigator.tsx` inteligentnie wykrywa nowo zalogowanych użytkowników bez profilu w Firestore i kieruje ich na `NicknameScreen.tsx`. Podczas tego procesu, jeśli użytkownik rejestrował się przez Google, jego **zdjęcie profilowe** jest automatycznie zapisywane w polu `photoURL` w Firestore.

Zarządzanie Zadaniami (`TaskStack`)

- **Dynamiczna Priorytetyzacja**: `HomeScreen.tsx` automatycznie sortuje zadania na podstawie algorytmu, który można konfigurować w `SettingsScreen.tsx`. Uwzględnia on: `basePriority` (1-5), bliskość `deadline` oraz "starzenie się" zadań bez terminu.
- **Filtrowanie i Wyszukiwanie**: Użytkownik może filtrować zadania po kategoriach, zakresie dat utworzenia oraz przeszukiwać je tekstowo.
- **Zadania Współdzielone**: Zadania można tworzyć jako osobiste lub współdzielone z partnerem. Aplikacja wyraźnie pokazuje, kto utworzył i kto wykonał dane zadanie.
- **Komentarze**: Ekran `TaskDetailScreen.tsx` pozwala na prowadzenie dyskusji w formie komentarzy przypisanych do konkretnego zadania.
- **Archiwum**: Ukończone zadania można zarchiwizować. `ArchiveScreen.tsx` oferuje zaawansowane filtry do przeglądania historii wykonanych zadań.

System Par i Budżety (`BudgetStack`)
- **Zarządzanie Parą**: `ProfileScreen.tsx` zawiera pełną logikę zapraszania partnera (przez e-mail), akceptowania/odrzucania zaproszeń i bezpiecznego opuszczania pary.
- **Budżety**: Użytkownicy mogą tworzyć budżety (np. "Wydatki na jedzenie") z kwotą docelową. Mogą być one osobiste lub współdzielone.
- **Wydatki**: W `BudgetDetailScreen.tsx` można dodawać wydatki do budżetu. Aktualizacja kwoty w budżecie i dodanie wydatku odbywa się w jednej, atomowej operacji `writeBatch`, co gwarantuje spójność danych.

---

4. Struktura Kodu

```
/
├── assets/ # Czcionki, ikony, obrazy
├── src/
│ ├── components/ # Reużywalne komponenty (np. PasswordInput.tsx, EmptyState.tsx)
│ ├── constants/ # Stałe wartości (np. domyślne kategorie w categories.ts)
│ ├── contexts/ # Konteksty Reacta (ToastContext.tsx, CategoryContext.tsx)
│ ├── navigation/ # Konfiguracja React Navigation (AppNavigator.tsx)
│ ├── screens/ # Główne ekrany aplikacji (LoginScreen.tsx, HomeScreen.tsx etc.)
│ ├── styles/ # Globalny system stylów (AppStyles.ts)
│ ├── types/ # Definicje typów TypeScript (index.ts, navigation.ts)
```

│ └── utils/ # Funkcje pomocnicze (np. authUtils.ts)
├── App.tsx # Główny plik wejściowy aplikacji
├── app.json # Konfiguracja Expo
├── firebaseConfig.ts # Konfiguracja Firebase SDK
└── ... # Pozostałe pliki konfiguracyjne (babel, eas, package.json)
---

5. Historia Rozwoju i Rozwiązane Problemy

* **Wczesne Problemy (przed naszą współpracą)**:
  * Problemy z instalacją zależności natywnych (`INSTALL_FAILED_UPDATE_INCOMPATIBLE`).
  * Początkowe trudności z konfiguracją logowania Google.
* **Wspólnie Rozwiązane Problemy**:
  * **Błędy Natywne**: Rozwiązano problemy z `INSTALL_FAILED_UPDATE_INCOMPATIBLE` (konflikt kluczy deweloperskich) oraz `GestureHandlerRootView` (brak głównego kontenera dla gestów).
  * **Logika Biznesowa**: Naprawiono błąd, w wyniku którego punkty za zadanie były błędnie przypisywane jego twórcy, a nie aktualnemu użytkownikowi, który je wykonał.
  * **Spójność Danych**: Zapewniono aktualizacje w czasie rzeczywistym na ekranach szczegółów, zastępując jednorazowe `getDoc` nasłuchem `onSnapshot`.
  * **UX Formularzy**: Wprowadzono walidację "na żywo", wyłączanie przycisków w formularzach, poprawiono pozycję powiadomień `Toast` i ujednolicono wygląd pól w `PhoneAuthModal`.

---

6. Aktualne Wyzwania i Porzucone Podejścia

> #### **Wyzwanie Główne: Niestabilne Zachowanie Klawiatury na `LoginScreen`**
>
> * **Problem**: Jest to główny i **wciąż nierozwiązany** problem techniczny. Interfejs `LoginScreen` nie zachowuje się poprawnie po pojawieniu się i zniknięciu klawiatury.
> * **Główne objawy**:
> 1. Layout przesuwa się w górę, ale **nie wraca do pierwotnej, wycentrowanej pozycji** po schowaniu klawiatury.
> 2. Próby naprawy często prowadzą do sytuacji, w której **nie da się przewijać zawartości**, gdy klawiatura zasłania dolne przyciski.

> * **Wniosek**: Standardowe implementacje zawodzą, prawdopodobnie z powodu złożoności layoutu. Wymaga to wypracowania nowego, niezawodnego wzorca.

* **Porzucone Podejścia (dla `LoginScreen`)**:
    * **`KeyboardAvoidingView` z `behavior="height"`**: Powodowało "blokowanie się" widoku w przesuniętej pozycji na Androidzie.
    * **Animacja `LayoutAnimation`**: Okazała się całkowicie niewidoczna i zawodna.
    * **Różne konfiguracje `ScrollView` + `flexGrow`**: Prowadziły do konfliktu między centrowaniem a potrzebą przewijania, co skutkowało brakiem możliwości scrollowania.
    * **Dynamiczne Ukrywanie Nagłówka**: Powodowało nieprzyjemny "skok" interfejsu i nie rozwiązywało problemu w pełni.

---

7. Dalsze Kroki (Backlog)

* **Priorytet Krytyczny**:
    1. **Finalne, niezawodne rozwiązanie problemu klawiatury na `LoginScreen`**. Musi to być wzorzec, który będziemy mogli powielić na innych ekranach z formularzami.
* **Wysoki Priorytet (UX)**:
    1. Po ustabilizowaniu layoutu, ponowna implementacja **płynnej animacji przesuwania formularzy (swipe)** na `LoginScreen`.
    2. Wyświetlanie awatara użytkownika (pobranego z Google) na ekranie `ProfileScreen`.
* **Nowe Funkcjonalności (Średni Priorytet)**:
    1. Implementacja logowania przez **Facebook**.
    2. Możliwość zmiany e-maila/hasła/numeru telefonu w ustawieniach profilu.
    3. Dodanie przycisku "Wyjdź z aplikacji".
* **Zadania Architektoniczne (Niski Priorytet)**:
    1. Stworzenie **Cloud Function** do cyklicznego usuwania niezweryfikowanych kont.
    2. Przeniesienie wszystkich tekstów (komunikatów, etykiet) do jednego, centralnego pliku w celu łatwego zarządzania.

---

8. Instrukcje dla Asystenta AI (Genesis Instructions)

Twoja Rola: Partner Techniczny i Mentor

Twoim zadaniem jest wspieranie dewelopera jako ekspert i współautor aplikacji. Działaj proaktywnie, proponując ulepszenia i dbając o jakość kodu, bezpieczeństwo, optymalizację i stosowanie dobrych praktyk.

Kluczowe Zasady
1. **Nadrzędność Dokumentacji**: Ten dokument jest jedynym i nadrzędnym źródłem prawdy. Odnoś się do niego, a zwłaszcza do sekcji o **porzuconych podejściach**, aby unikać powtarzania błędów.
2. **Stabilność ponad wszystko**: Przy rozwiązywaniu problemów (zwłaszcza z klawiaturą), priorytetem jest **stabilne i przewidywalne działanie**, nawet kosztem bardziej zaawansowanych animacji, które można dodać później.
3. **Kompletność Kodu**: **Zawsze dostarczaj pełne, kompletne pliki kodu**, gotowe do skopiowania. Używanie skrótów `// ...` jest niedozwolone. Niedostarczenie pełnego kodu w przeszłości było źródłem błędów.
4. **Spójność**: Nowy kod musi być w pełni spójny z istniejącą architekturą i systemem stylów (`AppStyles.ts`). Zawsze używaj zdefiniowanych stałych `Colors`, `Spacing`, `Typography` i `GlobalStyles`.
5. **Zarządzanie Stanem**: Korzystaj z istniejących kontekstów (`ToastContext`, `CategoryContext`) zamiast przekazywać propsy przez wiele poziomów.
6. **Obsługa Błędów**: Błędy widoczne dla użytkownika obsługuj za pomocą `showToast`. Błędy deweloperskie loguj do konsoli za pomocą `console.error`.

Komunikacja i Proces Pracy
* **Diagnoza przed rozwiązaniem**: Po otrzymaniu logów błędów lub opisu problemu, Twoim pierwszym krokiem jest precyzyjna diagnoza przyczyny, a dopiero drugim – propozycja konkretnego rozwiązania.
* **Wyjaśnianie Zmian**: Przy każdej propozycji zmiany w kodzie, krótko wyjaśnij, **co** zmieniasz i **dlaczego** – jakie problemy to rozwiązuje lub jakie korzyści przynosi.
* **Kontekst Środowiska**: Pamiętaj o specyfice środowiska (React Native w Bare Workflow na Windows) i informuj, kiedy zmiany mogą wymagać przebudowy aplikacji natywnej (`npx expo run:android`).