Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Oliver Gruba, Maciej Nasiadka

26 stycznia 2026

Imię i Nazwisko (nr indeksu)	Oliver Gruba (292583) Maciej Nasiadka (292574)
Nazwa uczelni	Uniwersytet Gdański
Kierunek	Informatyka (profil praktyczny)
Prowadzący	dr inż. Stanisław Witkowski
Temat	Dokumentacja techniczna szczegółowych warunków dla wykonania wybranego systemu informatycznego.
System	Kairo habit tracker (aplikacja do śledzenia nawyków)
Numer sprawozdania	12
Data zajęć	22.01.2026
Data oddania	28.01.2026
Miejsce na ocenę	

Spis treści

1. Podział systemów informatycznych

W literaturze wyróżnia się między innymi następujące kryteria podziału systemów informatycznych:

- **Ze względu na architekturę:** scentralizowane, klient–serwer, mikroserwisy, systemy rozproszone.
- **Ze względu na przeznaczenie:** transakcyjne (CRUD), analityczne (BI/Analytics), wspierające decyzje (DSS), systemy komunikacyjne.
- **Ze względu na kanał dostępu:** webowe, mobilne (native/cross–platform), desktopowe.
- **Ze względu na tryb pracy:** wsadowe, interaktywne (real–time), asynchroniczne (kolejki/cron).

Kairo klasyfikuje się jako:

- **System mobilny klient–serwer** (aplikacja mobilna React Native/Expo ↔ REST API Laravel).
- **System transakcyjno–analityczny** (rejestracja/logowanie, CRUD nawyków, agregacje postępu, osiągnięcia).
- **System rozproszony** z elementami asynchronicznymi (powiadomienia generowane po stronie backendu, zadania cykliczne).
- **Warstwowy** (Prezentacja, Logika, Dane, Infrastruktura/DevOps).

2. Cel systemu informatycznego

Kairo to aplikacja wspierająca budowanie i utrzymywanie dobrych nawyków. System pozwala:

- definiować nawyki (katalog wstępnie zdefiniowanych oraz własne),
- planować wykonania (dni tygodnia, godziny, okresy aktywności),
- otrzymywać **powiadomienia backendowe** przypominające o zaplanowanych aktywnościach,
- oznaczać wykonania i śledzić serię (*streak*),

- rywalizować i motywować się społecznie (zaproszenia do znajomych, lista znajomych),
- zdobywać osiągnięcia i przeglądać postęp.

2.1. Cele biznesowe (wg kryteriów SMART)

- **Rejestracje:** osiągnąć *1000* zarejestrowanych użytkowników w ciągu *3 miesięcy* od wdrożenia MVP (mierzalne w statystykach backendu).
- **Aktywność dzienna (DAU):** uzyskać 30% aktywnych użytkowników dziennie po *6 miesiącach* (mierzone na podstawie logowań i wywołań API).
- **Retencja miesięczna:** utrzymać 40% użytkowników powracających w kolejnym miesiącu (analiza cohort w Analytics).
- **Konwersja na subskrypcję:** osiągnąć 5% konwersji do planu premium w ciągu 6 miesięcy (dane z tabeli `subscriptions`).
- **Skuteczność powiadomień:** 70% zaplanowanych przypomnień skutkuje akcją użytkownika w ciągu 2 godzin (eventy wykonania nawyku po wysłaniu powiadomienia).

2.2. Cele techniczne

- **Dostępność API:** 99,9% miesięcznie (monitoring, healthcheck, redundancja).
- **Wydajność:** czas odpowiedzi p95 300 ms dla kluczowych endpointów (habits, profile, achievements).
- **Bezpieczeństwo:** uwierzytelnianie **Sanctum** z tokenami i weryfikacją e–mail; hasła hashowane, komunikacja *HTTPS*.
- **Skalowalność:** poziome skalowanie aplikacji (kontenery), niezależna skala bazy MySQL/MariaDB.
- **Obsługa powiadomień:** backendowy harmonogram i wysyłka (kolejki/cron, Laravel Notifications, ewentualnie FCM/APNs).
- **Jakość:** testy Feature, CI/CD, statyczna analiza (PHPStan, ESLint), logowanie i obserwowalność.

2.3. Grupa docelowa

- Osoby chcące wdrażać i utrzymywać codzienne nawyki (studenci, młodzi profesjonaliści, osoby w procesie rozwojowym).
- Użytkownicy mobilni (Android/iOS) preferujący prostą, czytelną aplikację z przypomnieniami i przejrzystą analityką.
- Użytkownicy ceniący motywację społeczną (lista znajomych, zaproszenia) i elementy grywalizacji (osiągnięcia, streak).

3. Wymagania funkcjonalne i niefunkcjonalne systemu

Analiza wymagań została przeprowadzona w oparciu o wywiady z potencjalnymi użytkownikami oraz analizę konkurencyjnych rozwiązań rynkowych.

3.1. Wymagania funkcjonalne

Wymagania funkcjonalne opisują zachowanie systemu w konkretnych scenariuszach użycia.

- **Uwierzytelnianie i konto:** rejestracja, logowanie, wylogowanie (dla sesji bieżącej i wszystkich), weryfikacja adresu e–mail, reset hasła.
- **Profil użytkownika:** podgląd profilu, edycja nazwy wyświetlanej, zarządzanie avatarem (upload/usuwanie).
- **Nawyki (katalog i własne):** lista wbudowanych nawyków, tworzenie/edycja/usuwanie własnych (`habits/custom`), powiązanie z użytkownikiem (`user_habits`).
- **Planowanie i przypomnienia:** dni tygodnia, godzina, daty start/koniec; **powiadomienia generowane przez backend** zgodnie z konfiguracją nawyku.
- **Wykonania nawyku:** oznaczanie *complete/uncomplete*, śledzenie *streak*, historia wykonań.
- **Osiągnięcia:** przypisanie do użytkownika, prezentacja stanu odblokowania.
- **Relacje społeczne:** zaproszenia do znajomych (wysłane/otrzymane, akceptacja/odrzucenie), lista znajomych, usuwanie z listy.
- **Subskrypcje:** podgląd stanu subskrypcji (integracja ze Stripe).
- **Analityka i postęp:** agregacja statystyk (np. liczba wykonań, najlepszy streak), przegląd w aplikacji mobilnej.

3.2. Wymagania niefunkcjonalne

- **Bezpieczeństwo:** tokeny dostępu (Sanctum), kontrola dostępu, weryfikacja e–mail, szyfrowanie haseł, komunikacja HTTPS.
- **Wydajność i skalowalność:** krótki czas odpowiedzi, możliwość skalowania poziomego, indeksy w bazie dla kluczowych kolumn (`user_id`, `habit_id`, `created_at`).
- **Niezawodność:** odporność na błędy, monitorowanie, logowanie zdarzeń, automatyczne odtwarzanie w razie awarii.
- **Użyteczność:** prosta i spójna nawigacja, czytelne komunikaty błędów, dostępność na Android/iOS.
- **Utrzymywalność:** podział na warstwy, czytelne API, testy automatyczne, CI/CD.

4. Propozycja wyboru narzędzi i platform do realizacji systemu

4.1. Warstwa Prezentacji (Frontend)

- **Technologia:** React Native (Expo) z `expo-router`, stylowanie z `Tamagui`.
- **Zarządzanie stanem:** konteksty `AuthContext`, `HabitsContext`; hooki (np. lokalne powiadomienia, znajomi, osiągnięcia).
- **Komunikacja z API:** własny wrapper `apiFetch` z nagłówkiem `Authorization: Bearer <token>`.
- **Przechowywanie:** `SecureStore`/`AsyncStorage` dla tokenów i drobnych danych lokalnych.

4.2. Warstwa Logiki (Backend)

- **Technologia:** Laravel (PHP) z `Sanctum` do tokenów API i `Notifications` do wysyłki powiadomień.
- **API REST:** moduły: `auth`, `email`, `profile`, `habits/custom`, `habits/user`, `friend-requests`, `friends`, `achievements`, `subscription`.
- **Powiadomienia: implementowane po stronie backendu** (cron/kolejki; kanały mail/push). Backend zgodnie z konfiguracją nawyków generuje przypomnienia.
- **Walidacja i zasady:** reguły domenowe (np. `DaysOfWeek`), zasoby API (Resources) i kontrolery per moduł.

4.3. Warstwa Danych

- **Baza danych: MySQL (MariaDB)** – główne tabele: `users`, `user_infos`, `habits`, `user_habits`, `user_habit_completions`, `achievements`, `user_achievements`, `friend_requests`, `friends`, `subscriptions`, `personal_access_tokens`.
- **Indeksy:** kluczowe kolumny `user_id`, `habit_id`, `created_at`; relacje *FK* z kasowaniem kaskadowym.
- **Migracje:** pełna definicja schematu w katalogu `database/migrations`.

4.4. Infrastruktura i DevOps

- **Konteneryzacja:** `Dockerfile`, `docker-compose.yml`; rozdzielenie usług (backend, baza, cache, reverse proxy).
- **CI/CD:** GitHub Actions (workflow `deploy-vps.yml`); automatyzacja testów i wdrożeń.
- **Konfiguracja:** środowiska `.env`, logowanie i monitoring; automatyczne zadania (cron/kolejki) dla powiadomień.

```
-- users + user_infos (MariaDB)
CREATE TABLE users (
    id CHAR(26) PRIMARY KEY,       -- ULID
    email VARCHAR(255) UNIQUE NOT NULL,
    email_verified_at DATETIME NULL,
    password VARCHAR(255) NOT NULL,
    remember_token VARCHAR(100) NULL,
    created_at TIMESTAMP NULL,
    updated_at TIMESTAMP NULL
);

CREATE TABLE user_infos (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id CHAR(26) NOT NULL,
    name VARCHAR(255) NULL,
    avatar_url VARCHAR(512) NULL,
    coins INT DEFAULT 0,
    created_at TIMESTAMP NULL,
    updated_at TIMESTAMP NULL,
    CONSTRAINT fk_user_infos_user
        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
```

```
22 );
```

Listing 1: Przykladowa konfiguracja modelu danych uzytkownika

5. Wybór min. 7 diagramów - dla opisu założeń zadania

1. **Diagram Przypadków Użycia (Use Case Diagram) ref: 1:**

   Cel: Przedstawienie zakresu funkcjonalnego systemu z perspektywy aktorów zewnętrznych. Diagram identyfikuje główne interakcje użytkowników z aplikacją Kairo oraz pozwala jednoznacznie określić, które funkcjonalności są dostępne dla użytkownika niezarejestrowanego, a które wymagają uwierzytelnienia.

2. **Diagram Klas (Class Diagram) ref: 2:**

   Cel: Opis struktury logicznej systemu poprzez zdefiniowanie klas domenowych, ich atrybutów, metod oraz relacji pomiędzy nimi. Diagram stanowi podstawę do implementacji modelu danych oraz warstwy logiki biznesowej backendu.

3. **Diagram Sekwencji (Sequence Diagram) ref: 6.3:**

   Cel: Zobrazowanie dynamicznego przebiegu komunikacji pomiędzy aktorami, aplikacją mobilną i backendem w czasie realizacji wybranych scenariuszy (np. rejestracja, dodanie nawyku, wysłanie powiadomienia). Diagram pozwala przeanalizować kolejność wywołań API oraz przepływ danych.

4. **Diagram Aktywności (Activity Diagram) ref: 4:**

   Cel: Przedstawienie przepływu czynności (workflow) w kluczowych procesach systemowych, takich jak codzienne zarządzanie nawykami czy proces rejestracji użytkownika. Diagram uwzględnia decyzje, rozgałęzienia oraz możliwe ścieżki alternatywne.

5. **Diagram BPMN (Business Process Model and Notation) ref: 5:**

   Cel: Opis projektu pod względem przepływu zadań i budowy projektu dla biznesu. Pokazujący proces biznesowy od początku do końca

6. **Diagram Komponentów (Component Diagram) ref: 6:**

   Cel: Zaprezentowanie architektury logicznej systemu na poziomie komponentów (Frontend, API, moduły backendowe, baza danych, usługi powiadomień). Diagram ilustruje zależności pomiędzy komponentami oraz sposób ich integracji.

7. **Diagram Wdrożenia (Deployment Diagram) ref: 7:**

Cel: Przedstawienie fizycznego rozmieszczenia elementów systemu w środowisku uruchomieniowym. Diagram pokazuje, na jakich węzłach (serwer, kontenery, urządzenie mobilne) działają poszczególne komponenty oraz jak odbywa się komunikacja sieciowa.

Powyższy zestaw diagramów zapewnia kompletność dokumentacji technicznej, pokrywając widok 4+1 architektury oprogramowania.

6. Diagramy UML wraz z opisem ich zastosowania w projekcie

W tej sekcji przedstawiono szczegółowe projekty diagramów wraz z analizą.

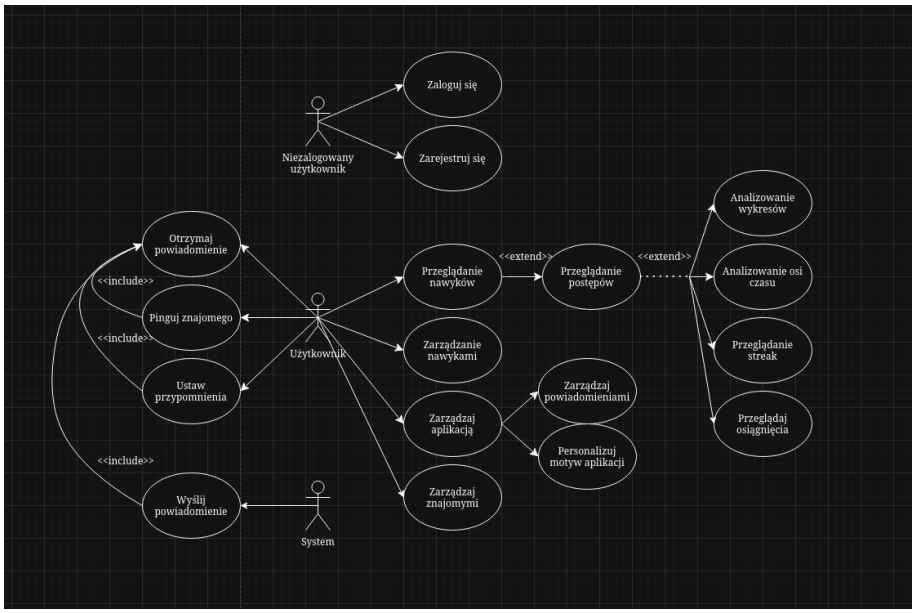
Weryfikacja zgodności z implementacją

Na podstawie aktualnego kodu (Frontend i Backend) potwierdzono zgodność następujących elementów i wskazano różnice wymagające korekty diagramów:

- **Zgodne:** *User*, *Habit*, *UserHabit*, *UserHabitCompletion*, *Achievement*, zaproszenia do znajomych (*FriendRequest*), moduły profilu, uwierzytelniania, osiągnięć i nawyków.
- **Różnice:** w implementacji relacje znajomości są przechowywane w tabeli `friends` (pivot), brak odrębnej klasy *Friendship*; zamiast klasy *Notification* używany jest mechanizm powiadomień Laravel (kanały mail/push), a konfiguracja przypomnień wynika z pól na *UserHabit* (`notification_time`, `days_of_week`). Klasa *UserSettings* nie występuje – ustawienia profilu są w `user_infos`.
- **Sekwencje/Activity/BPMN:** przepływy auth/profile/habits pokrywają się z trasami API; przepływ powiadomień należy powiązać z harmonogramem backendu zamiast wyłącznie lokalnych notyfikacji.

6.1. Diagram Przypadków Użycia (Use Case)

Diagram ten ilustruje główne funkcjonalności systemu dostępne dla poszczególnych aktorów. Ujmuje interakcje między użytkownikami i systemem, definiując zakres aplikacji z perspektywy użytkownika końcowego.



Rysunek 1: Diagram przypadków użycia Aplikacji Kairo Habit

Opis zastosowania:

Aktorzy systemu:

- **Użytkownik Niezarejestrowany** – osoba bez konta, która może jedynie się zarejestrować lub zalogować
- **Użytkownik Zarejestrowany** – osoba z aktywnym kontem mająca dostęp do pełnej funkcjonalności aplikacji
- **System** – backend aplikacji obsługujący automatyczne procesy (powiadomienia, obliczanie statystyk, harmonogramowanie przypomnień)
- **Przyjaciel** – inny użytkownik w sieci społecznej danego użytkownika

Główne przypadki użycia (UC):

1. **UC1: Rejestracja nowego konta** – Użytkownik niezarejestrowany tworzy konto podając email i hasło. System waliduje unikalność emaila (sprawdzenie w bazie danych), haszuje hasło oraz tworzy rekord użytkownika.
2. **UC2: Logowanie** – Zarejestrowany użytkownik loguje się przez podanie emaila i hasła. Backend waliduje poświadczenia, generuje token sesji (JWT) i zwraca token do aplikacji mobilnej.
3. **UC3: Dodanie nowego nawyku** – Zalogowany użytkownik definiuje nowy nawyk (nazwa, opis, częstotliwość wykonania, opcjonalne przypomnienie). System zapisuje nawyk w bazie danych i aktywuje harmonogram przypomnień.

4. **UC4: Oznaczenie nawyku jako ukończony** – Użytkownik zaznacza wykonanie nawyku w danym dniu. System rejestruje wykonanie w tabeli `habit_executions`, aktualizuje licznik serii (streak) i statystyki.

5. **UC5: Przeglądanie postępu** – Użytkownik przegląda swoje statystyki, wykresy i osiągnięcia. Backend agreguje dane z tabeli `habit_executions`, oblicza procent ukończenia, liczę serii i generuje dane dla visualizacji.

6. **UC6: Dodanie przyjaciela** – Zalogowany użytkownik wysyła zaproszenie do innego użytkownika. System tworzy wpis w tabeli `friendships` ze statusem "pending"i wysyła powiadomienie.

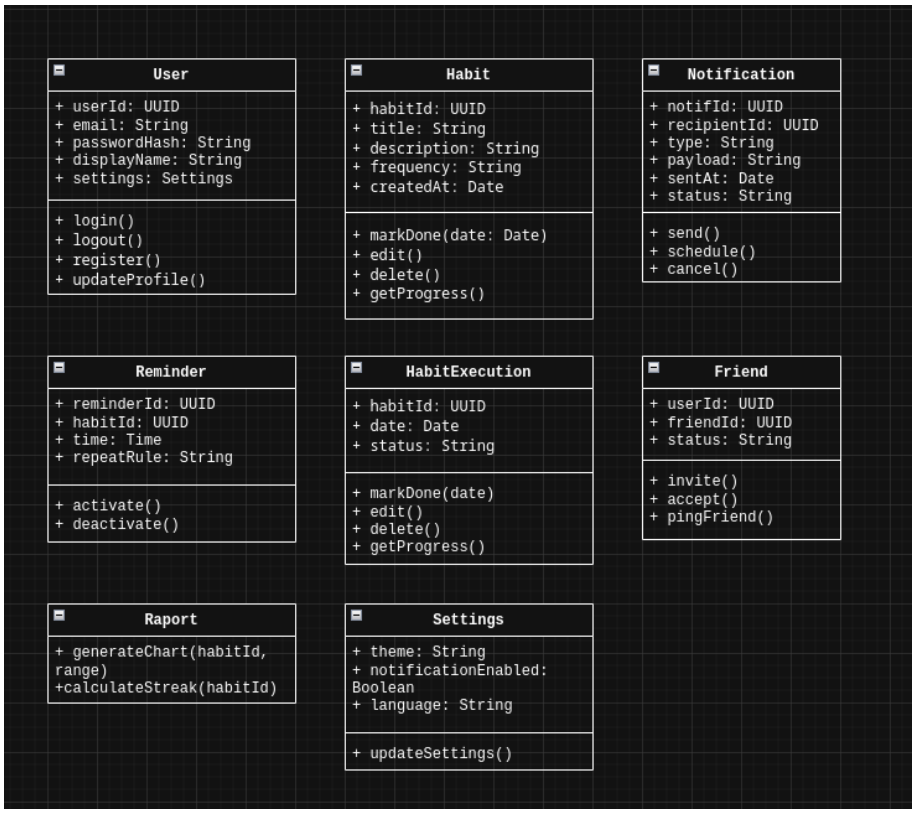
7. **UC7: Wysłanie motywacyjnego ping'a** – Użytkownik wysyła powiadomienie motywacyjne do przyjaciela. Backend rejestruje powiadomienie w tabeli `notifications` i wysyła push notification.

8. **UC8: Zarządzanie ustawieniami** – Użytkownik edytuje profil (nazwę wyświetlaną), ustawienia powiadomień, motyw aplikacji i strefę czasową. System aktualizuje wpisy w tabelach `users` i `user_settings`.

Relacje między use case'ami:

- UC2 (Logowanie) jest *extend* dla UC1 – możliwość logowania pojawia się po rejestracji
- UC3-UC7 są *include* dla UC2 – wszystkie te funkcje wymagają zalogowania
- UC5 (Przeglądanie postępu) zależy od UC4 – statystyki wymagają wcześniejszych wykonań

6.2. Diagram Klas (Class Diagram)



Rysunek 2: Diagram klas użycia Aplikacji Kairo Habit

Opis zastosowania:

Diagram klas definiuje strukturę danych i logikę biznesową aplikacji. Reprezentuje osiem kluczowych klas, ich atrybuty, metody oraz relacje pomiędzy nimi.

Klasy podstawowe:

1. Klasa User (Użytkownik)

```
class User {
  // Atrybuty
  userId: UUID // Unikalny identyfikator
  email: String // Email (unikatowy)
  passwordHash: String // Zahaszowane haslo
  displayName: String // Nazwa wyswietlana
  createdAt: DateTime // Data zalozenia konta
  timezone: String // Strefa czasowa uzytkownika

  // Metody
  register(email, password): Boolean
  login(email, password): JWT
  updateProfile(displayName, timezone): void
  logout(): void
```

```
}
```

Listing 2: Struktura klasy User

Relacje: User posiada wiele Habits (1:*), jest uczestnikiem Friendships (relacja *:*), posiada Settings (1:1).

2. Klasa Habit (Nawyk)

```
class Habit {
  habitId: UUID
  userId: UUID (FK) // Klucz obcy do User
  title: String // Nazwa nawyku (np. "Czytanie")
  description: String
  frequency: Enum // DAILY, WEEKLY, CUSTOM
  color: String // Kolor wykresu (hex)
  icon: String // Ikona nawyku
  createdAt: DateTime

  // Metody
  markCompleted(date): void
  edit(title, description): void
  delete(): void
  getStreak(): Integer // Zwraca licze dni z rzedu
  getProgress(): Float // Procent ukonczenia
}
```

Listing 3: Struktura klasy Habit

Relacje: Habit należy do User (N:1), zawiera wiele HabitExecutions (1:*), posiada wiele Reminders (1:*), przydzielone Achievements (1:*).

3. Klasa HabitExecution (Log wykonania)

```
class HabitExecution {
  executionId: UUID
  habitId: UUID (FK)
  date: Date // Data wykonania
  status: Enum // COMPLETED, SKIPPED, MISSED
  timestamp: DateTime

  markAsCompleted(): void
  markAsSkipped(): void
}
```

Listing 4: Struktura klasy HabitExecution

Relacje: Reprezentuje kompozycję z Habit – wiele wykonań na nawyk.

4. Klasa Reminder (Przypomnienie)

```
class Reminder {
  reminderId: UUID
  habitId: UUID (FK)
  time: Time // Godzina przypomnienia
  repeatRule: String // DAILY, WEEKDAYS, CUSTOM
  enabled: Boolean
  timezone: String

  activate(): void
  deactivate(): void
  schedule(): void
}
```

Listing 5: Struktura klasy Reminder

Relacje: Przypomnienie należy do Habit (N:1).

5. Klasa Notification (Powiadomienie)

```
class Notification {
  notificationId: UUID
  recipientId: UUID (FK)
  type: Enum // REMINDER, FRIEND_PING, ACHIEVEMENT
  payload: JSON // Dane powiadomienia
  sentAt: DateTime
  readAt: DateTime (nullable)
  status: Enum // PENDING, SENT, READ

  send(): void
  markAsRead(): void
}
```

Listing 6: Struktura klasy Notification

Relacje: Powiadomienie jest adresowane do User (N:1).

6. Klasa Friendship (Przyjaźń)

```
class Friendship {
  friendshipId: UUID
  userId1: UUID (FK)
  userId2: UUID (FK)
  status: Enum // PENDING, ACCEPTED, REJECTED
  createdAt: DateTime

  invite(): void
  accept(): void
  reject(): void
  remove(): void
```

```
}
```

Listing 7: Struktura klasy Friendship

Relacje: Reprezentuje relację many-to-many między Users.

7. Klasa Achievement (Osiągnięcie)

```
class Achievement {
  achievementId: UUID
  habitId: UUID (FK)
  userId: UUID (FK)
  name: String // "7-dniowa seria", "30-dniowy streaker"
  description: String
  unlockedAt: DateTime
  icon: String

  unlock(): void
}
```

Listing 8: Struktura klasy Achievement

Relacje: Osiągnięcie przypisane do User i Habit (N:1 dla każdego).

8. Klasa UserSettings (Ustawienia)

```
class UserSettings {
  settingsId: UUID
  userId: UUID (FK)
  theme: Enum // LIGHT, DARK
  notificationsEnabled: Boolean
  language: String // "pl", "en"
  timezone: String

  updateSettings(theme, notifications): void
  getSettings(): UserSettings
}
```

Listing 9: Struktura klasy UserSettings

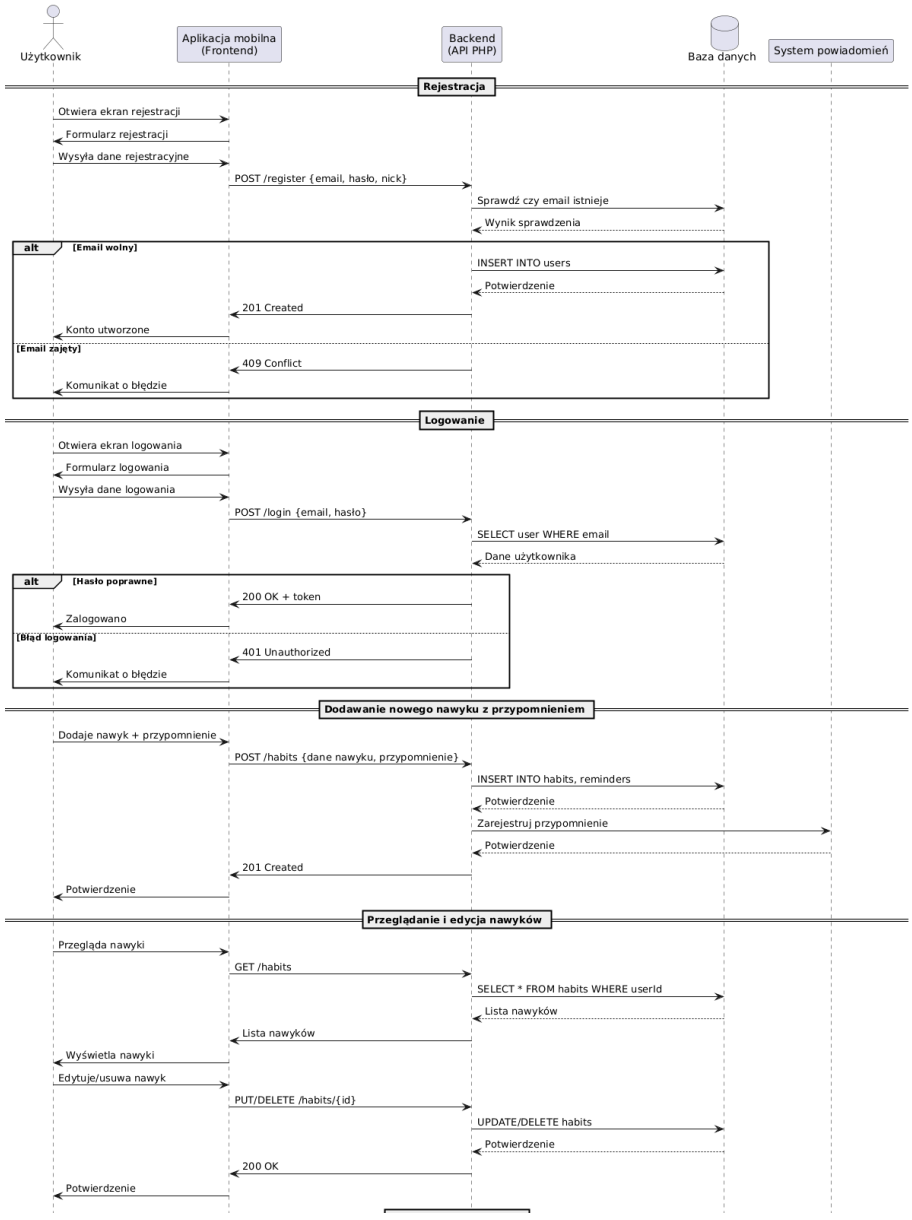
Relacje: One-to-one z User.

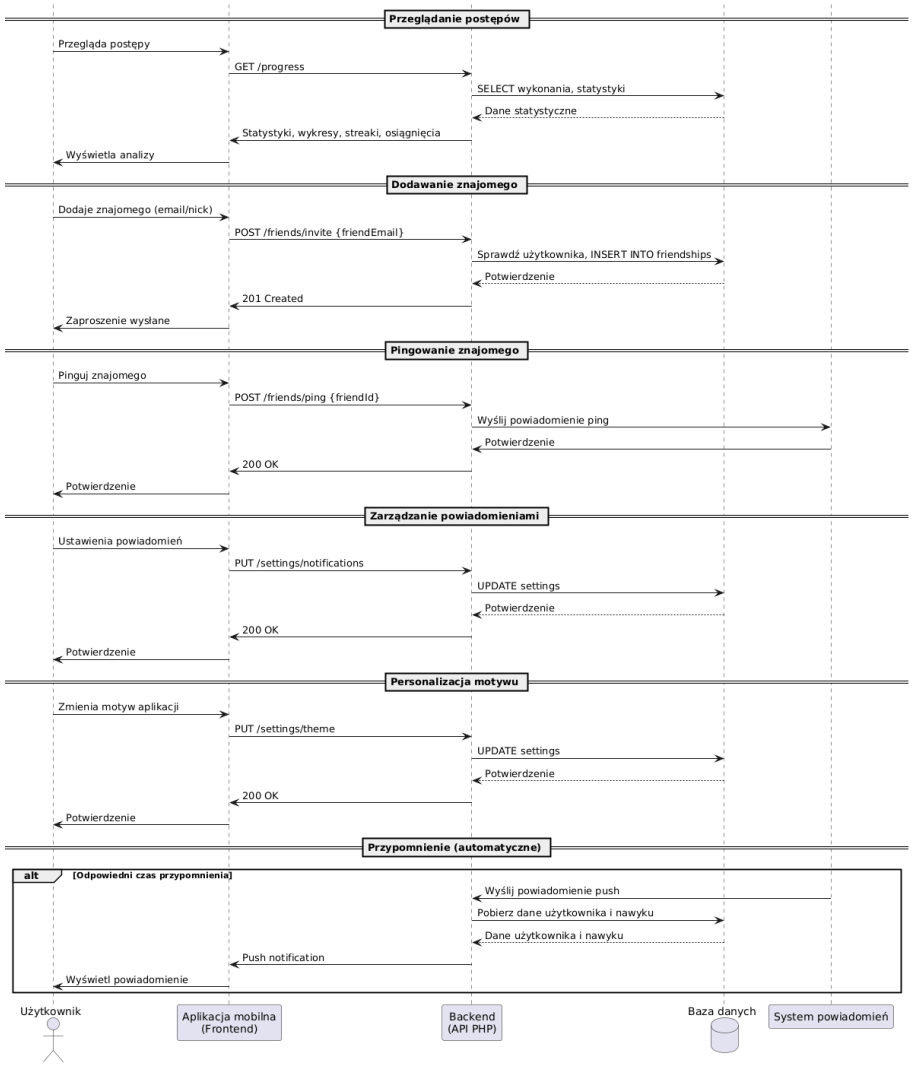
Podsumowanie relacji:

- User 1:* Habit – użytkownik ma wiele nawyków
- Habit 1:* HabitExecution – nawyk zawiera logów wykonań (kompozycja)
- Habit 1:* Reminder – nawyk może mieć wiele przypomnień
- Habit 1:* Achievement – osiągnięcia przypisane do nawyków

- User *:* User – przyjaźnie między użytkownikami (przez Friendship)
- User 1:1 UserSettings – każdy użytkownik ma ustawienia
- User 1:* Notification – użytkownik otrzymuje wiele powiadomień

6.3. Diagram Sekwencji (Sequence Diagram)





Rysunek 3: Diagram sekwencji aplikacji Kairo Habit

Opis przebiegu:

Diagram sekwencji pokazuje przepływ komunikacji pomiędzy aktorami a systemem dla trzech kluczowych scenariuszy.

Scenariusz 1: Rejestracja i Logowanie

1. **User** wypełnia formularz rejestracji (email, hasło)
2. **Mobile App** wysyła POST request: `POST /api/v1/users/register` z payloadsem:

```
{\"email\": \"user@example.com\", \"password\": \"secure123\"}
```

Listing 10: Request rejestracji

3. **API Backend** otrzymuje request, haszuje hasło (np. bcrypt)
4. **Backend** wysyła zapytanie do **Database**: INSERT do tabeli `users`

```
INSERT INTO users (userId, email, passwordHash, displayName,
    createdAt)
VALUES (UUID(), 'user@example.com', hash('secure123'), '
    user@example', NOW())
```

Listing 11: SQL INSERT

5. **Database** zwraca potwierdzenie (201 Created) lub błąd (409 Conflict jeśli email istnieje)
6. **Backend** zwraca response z user ID
7. **Mobile App** wyświetla komunikat sukcesu i przenosi do logowania
8. Podobny przepływ dla UC: Logowania – Backend generuje JWT token

Scenariusz 2: Dodanie Nawyku z Przypomnieniem

1. **User** (zalogowany) wypełnia formularz dodawania nawyku
2. **Mobile App** wysyła POST: `POST /api/v1/habits` z authenticacją (JWT header)

```
Headers: {Authorization: \"Bearer <JWT_TOKEN>\"}
Body: {
  \"title\": \"Czytanie\",
  \"description\": \"30 minut codziennie\",
  \"frequency\": \"DAILY\",
  \"reminderTime\": \"19:00\",
  \"timezone\": \"Europe/Warsaw\"
}
```

Listing 12: Request dodania nawyku

3. **Backend** waliduje token JWT (sprawdzenie ważności)
4. **Backend** executes INSERT do `habits` tabeli
5. **Backend** rejestruje **Reminder** w tabeli `reminders` z czasem `19:00`
6. **Notification Service** (lub Job Scheduler) otrzymuje event: "Habit created with reminder"
7. **Notification Service** harmonizuje przypomnienie w systemie (cron job lub message queue)
8. **Backend** zwraca 201 Created z nowym habitId
9. **Mobile App** wyświetla nawyk na liście

Scenariusz 3: Wysłanie Ping'a do Przyjaciela

1. **User A** wybiera przyjaciela z listy
2. **Mobile App** wysyła POST: `POST /api/v1/friends/{friendId}/ping`

```
Body: {\"message\": \"Keep going! You can do it!\"}
```

Listing 13: Request wyslania ping

3. **Backend** waliduje autentyczność (token JWT) i istnienie przyjaźni
4. **Backend** tworzy rekord w tabeli `notifications`:

```
INSERT INTO notifications (notificationId, recipientId, type,
    payload, sentAt, status)
VALUES (UUID(), <friendId>, 'FRIEND_PING', {...message...}, NOW(),
    'PENDING')
```

Listing 14: SQL INSERT notification

5. **Push Notification Service** (Firebase Cloud Messaging / APNs) wysyła powiadomienie push na urządzenie User B
6. **User B's Device** otrzymuje notification i wyświetla alert
7. **Backend** aktualizuje status powiadomienia na 'SENT' lub 'DELIVERED'
8. **User A's App** wyświetla potwierdzenie: "Ping sent!"

Scenariusz 4: Przeglądanie Postępu

1. **User** przechodzi do sekcji Analytics/Progress
2. **Mobile App** wysyła GET: `GET /api/v1/progress?month=2026-01`
3. **Backend** przetwarza query:

```
SELECT h.habitId, h.title, COUNT(CASE WHEN he.status='COMPLETED')
    as completed,
        COUNT(*) as total
FROM habits h
LEFT JOIN habit_executions he ON h.habitId = he.habitId
        AND DATE(he.date) BETWEEN '2026-01-01' AND '2026-01-31'
WHERE h.userId = <userId>
GROUP BY h.habitId
```

Listing 15: SQL agregacja statystyk

4. **Backend** oblicza dodatkowe metryki: streak (dni z rzędu), procent ukończenia

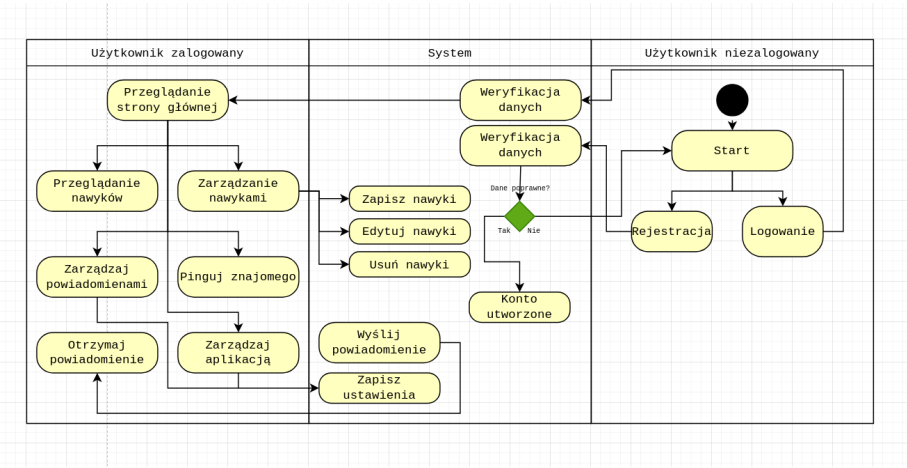
5. **Backend** zwraca JSON z danymi dla chart'ów:

```
{
  \"habits\": [
    {\"habitId\": \"123\", \"title\": \"Czytanie\", \"completed\":
       20, \"total\": 31,
     \"percentage\": 64.5, \"currentStreak\": 5}
  ]
}
```

Listing 16: Response progress

6. **Mobile App** renderuje wykresy i wizualizacje postępu

6.4. Diagram Aktywności (Activity Diagram)



Rysunek 4: Diagram aktywności Aplikacji Kairo Habit

Opis zastosowania:

Diagram aktywności pokazuje sekwencje czynności (workflow) dla głównych procesów użytkownika, uwzględniając decyzje i rozgałęzienia.

Proces 1: Rejestracja i Konfiguracja Profilu

1. **START** – Użytkownik otwiera aplikację

2. *Czynność:* Sprawdź czy użytkownik zalogowany?

3. *Decyzja:* [TAK] → Przejdź do dashboardu | [NIE] → Wyświetl ekran logowania

4. *Czynność:* Kliknij "Rejestracja"

5. *Czynność:* Wypełnij formularz (email, hasło, nazwa wyświetlana)
6. *Czynność:* Wyślij dane do backendu (POST /api/v1/users/register)
7. *Decyzja:* [Email dostępny] → Utwórz konto | [Email zajęty] → Pokaż błąd, wróć do kroku 3
8. *Czynność:* Zaloguj się automatycznie z wygenerowanym tokenem JWT
9. *Czynność:* Wyświetl questionnaire profilu (timezone, preferencje)
10. *Czynność:* Zapisz ustawienia w `user_settings` tabeli
11. **END** – Przejdź do dashboardu

Proces 2: Zarządzanie Nawykami (Daily Workflow)

1. **START** – Zalogowany użytkownik otwiera aplikację
2. *Czynność:* Wyświetl listę wszystkich nawyków (GET /api/v1/habits)
3. *Czynność:* Dla każdego nawyku, pokaż status dzisiaj (completed/pending)
4. *Decyzja:* [Użytkownik chce dodać nowy nawyk] → Przejdź do "Dodawanie nawyku [Chce oznaczyć jako wykonany] → Przejdź do "Oznaczanie nawyku"
5. *Czynność (Dodawanie):* Otwórz formularz dodawania
6. *Czynność:* Wprowadź: tytuł, opis, częstotliwość (DAILY/WEEKLY), opcjonalne przypomnienie
7. *Czynność:* POST do /api/v1/habits z danymi
8. *Czynność:* Backend INSERT do `habits`, następnie INSERT do `reminders` jeśli zadano czas
9. *Czynność (Oznaczanie):* Kliknij checkbox na nawyku
10. *Czynność:* POST /api/v1/habits/{habitId}/complete?date=today
11. *Czynność:* Backend INSERT do `habit_executions` (status='COMPLETED')
12. *Czynność:* Oblicz nową serię (streak) – SELECT COUNT() gdzie status='COMPLETED' AND consecutive days
13. *Decyzja:* [Streak osiągnął milestone: 7, 30, 100?] → Odblokuj Achievement | [Nie]
14. *Czynność:* Wyświetl wizualny feedback (checkmark, animation)

15. **END** – Powróć do listy nawyków

Proces 3: Interakcja ze Znajomymi

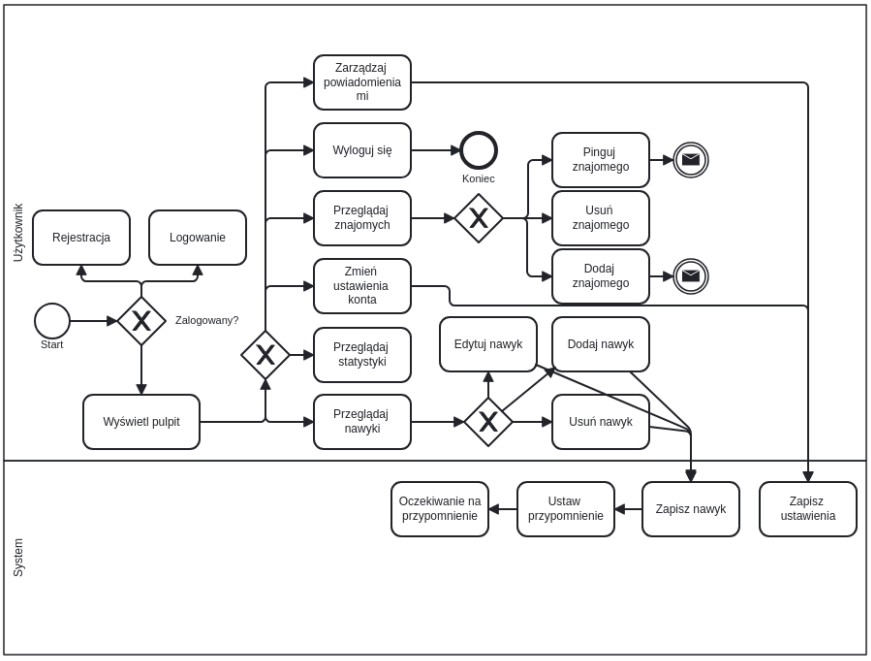
1. **START** – Zalogowany użytkownik przechodzi do sekcji "Przyjaciele"
2. *Czynność:* GET /api/v1/friends – pobierz listę zaakceptowanych przyjaciół
3. *Czynność:* Wyświetl listę z statusem (ostatnia aktywność, aktualna seria)
4. *Decyzja:* [Użytkownik chce dodać nowego przyjaciela] → Przejdź do "Zapraszania [Chce wysłać ping] → Przejdź do "Wysyłania ping'a"
5. *Czynność (Zapraszanie):* Wyszukaj użytkownika po emailu
6. *Czynność:* Kliknij "Wyślij zaproszenie"
7. *Czynność:* POST /api/v1/friendships/invite?targetUserId=XXX
8. *Czynność:* Backend INSERT do `friendships` (status='PENDING'), wysyła NOTIFICATION
9. *Decyzja:* [Przyjaciel zaakceptuje] → Aktualizuj status na ACCEPTED | [Odrzuci] → Usuń zaproszenie
10. *Czynność (Ping):* Wybierz przyjaciela z listy
11. *Czynność:* POST /api/v1/friends/{friendId}/ping z wiadomością motywacyjną
12. *Czynność:* Wyślij push notification (Firebase/APNs)
13. *Czynność:* Wyświetl potwierdzenie wysłania
14. **END** – Powróć do listy przyjaciół

Proces 4: Cykliczny Proces Przypomnień (System Job)

1. **START** – Scheduler odpalony (co godzinę lub w określonym interwale)
2. *Czynność:* Pobierz wszystkie aktywne przypomnienia: SELECT * FROM reminders WHERE enabled=true
3. **Dla każdego przypomnienia:**
4. *Czynność:* Sprawdź czy bieżący czas odpowiada czasowi przypomnienia (z uwzględnieniem timezone)

5. *Decyzja:* [Czas = reminder_time] → Przejdź dalej | [Time ≠ reminder_time] → Pomiń to przypomnienie
6. *Czynność:* Pobierz powiązany Habit i User
7. *Czynność:* Sprawdź czy dzisiejsze wykonanie już zalogowane: SELECT FROM habit_executions WHERE habitId=X AND date=TODAY
8. *Decyzja:* [Już wykonane] → Pomiń | [Nie wykonane] → Wyślij powiadomienie
9. *Czynność:* Wyślij push notification (tytuł: "Czas na: Habit.title")
10. *Czynność:* INSERT do `notifications` (type='REMINDER', status='SENT')
11. **END** – Czekaj na następny interval

6.5. Diagram BPMN



Rysunek 5: Diagram BPMN Aplikacji Kairo Habit

Opis zastosowania:

Diagram BPMN (Business Process Model and Notation) obrazuje procesy biznesowe aplikacji z podziałem na role (lanes) i przepływy decyzji (gateways). Diagram zawiera trzy główne pule (pools): użytkownik, system i usługa powiadomień.

Proces 1: Rejestracja i Logowanie (User Pool ↔ System Pool)

- **Start Event:** Użytkownik otwiera aplikację

- **Task (User):** Wypełnia formularz rejestracji
- **Task (System):** Waliduje email (SELECT COUNT(*) FROM users WHERE email=X)
- **Exclusive Gateway (Decyzja):**
    1. [Email dostępny] → Haszuj hasło (bcrypt), INSERT użytkownika
    2. [Email zajęty] → Zwróć błąd 409, powróć do Start
- **Task (System):** Generuj JWT token
- **Task (System):** INSERT domyślne ustawienia (user_settings)
- **Message Event:** Wyślij potwierdzenie emailem (opcjonalnie)
- **End Event:** Użytkownik zalogowany, przejście do dashboardu

Proces 2: Codzienne Wykonanie Nawyku (Daily Habit Execution)

- **Start Event (Timer):** Harmonogram przypomnień – godzina X dla użytkownika (z uwzględnieniem timezone)
- **Task (System):** Pobranie listy aktywnych nawyków z przypomnieniami
- **Task (Notification Service):** Wysłanie push notification na urządzenie użytkownika
- **Catch Event (Message):** Czeka na akcję użytkownika (max 24h timeout)
- **Exclusive Gateway (Decyzja):**
    1. [Użytkownik zaznaczył nawyk jako wykonany] Przejdź do "Rejestracji wykonania"
    2. [Użytkownik nie reaguje /odrzuca] Przejdź do "Logowania pominięcia"
- **Task (System - Rejestracja):** INSERT do habit_executions (status='COMPLETED'), UPDATE streak counter
- **Task (System - Logowanie pominięcia):** INSERT do habit_executions (status='SKIPPED' lub 'MISSED')
- **Exclusive Gateway (Sprawdzenie osiągnięcia):**
    1. [Streak=7 OR Streak=30 OR Streak=100] → Odblokuj Achievement

2. [Brak milestone'u] → Pomiń

- **Task (System):** Jeśli dostęp streakow, INSERT do achievements i wyślij notification gratulacyjne
- **End Event:** Proces nawyku zakończony do następnego dnia

Proces 3: Motywacja Społeczna (Friend Ping Flow)

- **Start Event (User A):** Użytkownik A inicjuje ping do przyjaciela B
- **Task (User A):** Wybiera przyjaciela z listy i wysyła wiadomość
- **Task (System):** Waliduje przyjaźń (SELECT FROM friendships WHERE userId1=A AND userId2=B AND status='ACCEPTED')
- **Exclusive Gateway:**
  1. [Przyjaźń istnieje] → Kontynuuj
  2. [Przyjaźń nie istnieje] → Zwróć błąd, koniec
- **Task (System):** INSERT do notifications (type='FRIEND_PING', recipientId=B)
- **Throw Event (Message):** Wyślij push notification do User B
- **Catch Event (User B):** User B otrzymuje powiadomienie na urządzeniu
- **Exclusive Gateway (Decyzja User B):**
  1. [Otwiera powiadomienie] → UPDATE notifications.readAt, pokaż motywacyjną wiadomość
  2. [Ignoruje powiadomienie] → Oznacz jako received
- **Task (System):** Wyślij feedback do User A: "Wiadomość przeczytana"lub "Wiadomość dostarczona"
- **End Event (User A):** Ping wysłany i potwierdzony

Proces 4: Agregacja Statystyk (Background Job - co godzinę)

- **Start Event (Timer):** Cron job, godzina X
- **Task (System):** SELECT DISTINCT userId FROM users WHERE isActive=true
- **Parallel Gateway:** Dla każdego użytkownika równolegle:
  - **Task:** Agregacja danych z habit_executions (procent ukończenia, streak)

  - **Task:** Aktualizacja statystyk w cache (Redis) lub tabeli summary
  - **Task:** Sprawdzenie czy są nowe osiągnięcia do odblokowania

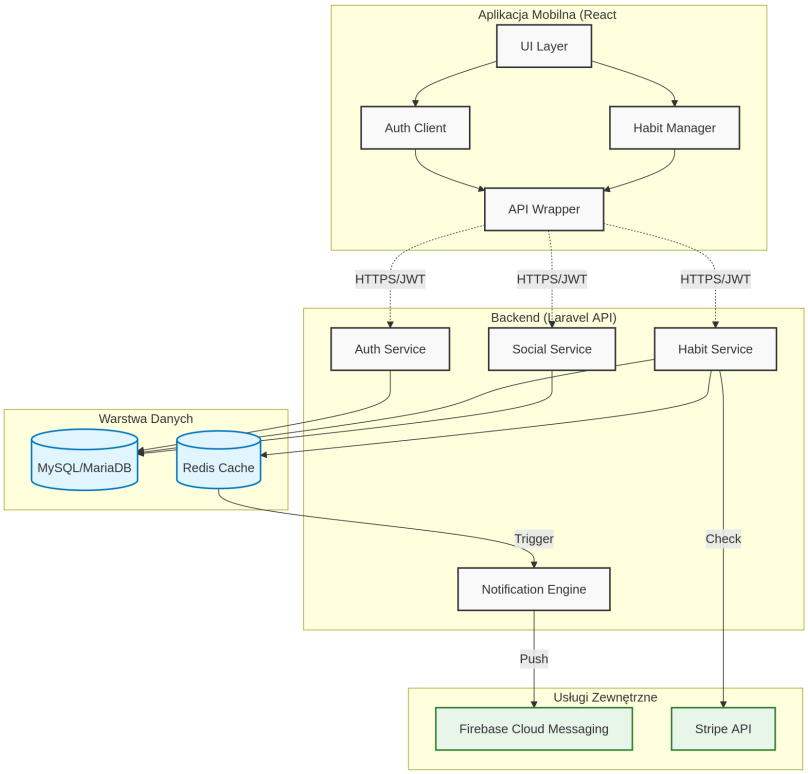
- **Synchronize Gateway:** Czeka aż wszystkie użytkownicy będą przetworzeni
- **End Event:** Statystyki zaktualizowane, gotowe do wyświetlenia

Elementy BPMN:

Element	**Znaczenie**
Koło (Event)	Rozpoczęcie, zakończenie, lub przerwanie procesu (Timer, Message)
Prostokąt (Task)	Konkretna czynność wykonywana przez użytkownika lub system
Diament (Gateway)	Bramka decyzji – rozgałęzienie oparte na warunku
Linia (Flow)	Przepływ kontroli między elementami
Swimlane (Lane)	Podział odpowiedzialności (User, System, Notification Service)

6.6. Diagram Komponentów (Component Diagram)

Przedstawia modułową budowę aplikacji, pokazując komponenty, interfejsy i ich zależności.



Rysunek 6: Diagram komponentów Aplikacji Kairo Habit

Opis zastosowania:

Architektura aplikacji Kairo opiera się na siedmiu kluczowych komponentach, które współpracują ze sobą poprzez dobrze zdefiniowane interfejsy.

1. Komponent "Authentication Äuthentication Authorization"Authorization"

- **Odpowiedzialność:** Zarządzanie rejestracją, logowaniem, weryfikacją tokenów JWT, kontrol dostępu (role-based)
- **Wejścia:** Poświadczenia użytkownika (email, hasło), token JWT
- **Wyjścia:** JWT token, user ID, role (USER, ADMIN)
- **Interfejs API:**

```
POST /api/v1/users/register
POST /api/v1/users/login
POST /api/v1/users/logout
GET /api/v1/users/me (verify token)
```

Listing 17: Auth API endpoints

- **Zależności:** Database (users, user_settings), Password Hashing Library

2. Komponent "Habit Management"

- **Odpowiedzialność:** CRUD operacje na nawyków, zarządzanie wykonaniami, obliczanie statystyk
- **Wejścia:** Dane nawyku (tytuł, opis, częstotliwość), user ID, data wykonania
- **Wyjścia:** Nawyk object, lista nawyków, statystyki (streak, completion
- **Interfejs API:**

```
GET /api/v1/habits (list all user habits)
POST /api/v1/habits (create habit)
GET /api/v1/habits/{habitId}
PUT /api/v1/habits/{habitId} (update)
DELETE /api/v1/habits/{habitId}
POST /api/v1/habits/{habitId}/complete (mark as done)
```

Listing 18: Habit API endpoints

- **Zależności:** Database (habits, habit_executions), Reminder Manager, Notification Component

3. Komponent "Reminder R̈eminder Scheduler"Scheduler"

- **Odpowiedzialność:** Zarządzanie harmonogramem przypomnień, triggerowanie notifications o zaplanowanym czasie
- **Wejścia:** Reminder details (time, frequency, enabled), habit ID, timezone
- **Wyjścia:** Notification event do Notification Component
- **Implementacja:** Cron job (node-cron lub APScheduler w Pythonie) lub Message Queue (RabbitMQ, Kafka)

```
setInterval(async () => {
  const reminders = await db.query(
    'SELECT * FROM reminders WHERE enabled=true AND time <= NOW()'
  );
  for (const reminder of reminders) {
    notificationComponent.sendReminder(reminder);
  }
}, 60000); // co minute
```

Listing 19: Reminder job pseudocode

- **Zależności:** Database (reminders, habits), Notification Component

4. Komponent "Notification Service"

- **Odpowiedzialność:** Wysyłanie push notifications, email notifications, przechowywanie logów powiadomień
- **Wejścia:** Notification payload (type, recipient, message), user device tokens
- **Wyjścia:** Potwierdzenie wysłania, status notifikacji (SENT, DELIVERED, READ)
- **Integracja:**
  - Firebase Cloud Messaging (FCM) dla urządzeń Android
  - Apple Push Notification (APNs) dla iOS
  - SendGrid/AWS SES dla email notifications

```
class NotificationService {
  async sendPushNotification(userId, message) {
    const deviceToken = await db.getUserDeviceToken(userId);
    const payload = {
      notification: { title: message.title, body: message.body },
      data: { habitId: message.habitId, type: message.type }
    };
    return firebase.messaging().sendToDevice(deviceToken, payload);
  }
}
```

Listing 20: Notification sending

- **Zależności:** Database (notifications), Firebase/APNs SDK

5. Komponent "Social Features"(Friends & Interactions)

- **Odpowiedzialność:** Zarządzanie przyjaźniami, wysyłanie ping'ów, wyświetlanie aktywności przyjaciół
- **Wejścia:** User ID, friend ID, ping message, friendship request
- **Wyjścia:** Friendship status, friend list, activity feed
- **Interfejs API:**

```
GET /api/v1/friends
POST /api/v1/friendships/invite?targetUserId=X
POST /api/v1/friends/{friendId}/ping
GET /api/v1/friends/{friendId}/activity
DELETE /api/v1/friendships/{friendshipId}
```

Listing 21: Social API endpoints

- **Zależności:** Database (friendships, friends_activity), Notification Component

6. Komponent "Analytics & Progress"

- **Odpowiedzialność:** Agregacja danych statystyk, obliczanie metryk, generowanie danych dla chart'ów
- **Wejścia:** User ID, date range, habit filters
- **Wyjścia:** Aggregated statistics (completion
- **Interfejs API:**

```
GET /api/v1/progress?month=2026-01&habitId=X
GET /api/v1/statistics/streaks
GET /api/v1/achievements
```

Listing 22: Analytics API

- **Implementacja:** Może korzystać z cache'u (Redis) dla szybkości

```
async getProgress(userId, month) {
  const cacheKey = `progress:${userId}:${month}`;
  let data = await redis.get(cacheKey);
  if (!data) {
    data = await db.query(`
      SELECT h.habitId, h.title,
             COUNT(CASE WHEN he.status='COMPLETED' THEN 1 END) as
                completed,
             COUNT(*) as total
      FROM habits h
      LEFT JOIN habit_executions he ON h.habitId = he.habitId
      WHERE h.userId = ? AND MONTH(he.date) = ?
      GROUP BY h.habitId
    `, [userId, month]);
    await redis.setex(cacheKey, 3600, JSON.stringify(data));
  }
  return data;
}
```

Listing 23: Analytics calculation

- **Zależności:** Database (habit_executions), Cache (Redis), Habit Management Component

7. Komponent "Data Access Layer"(Database)

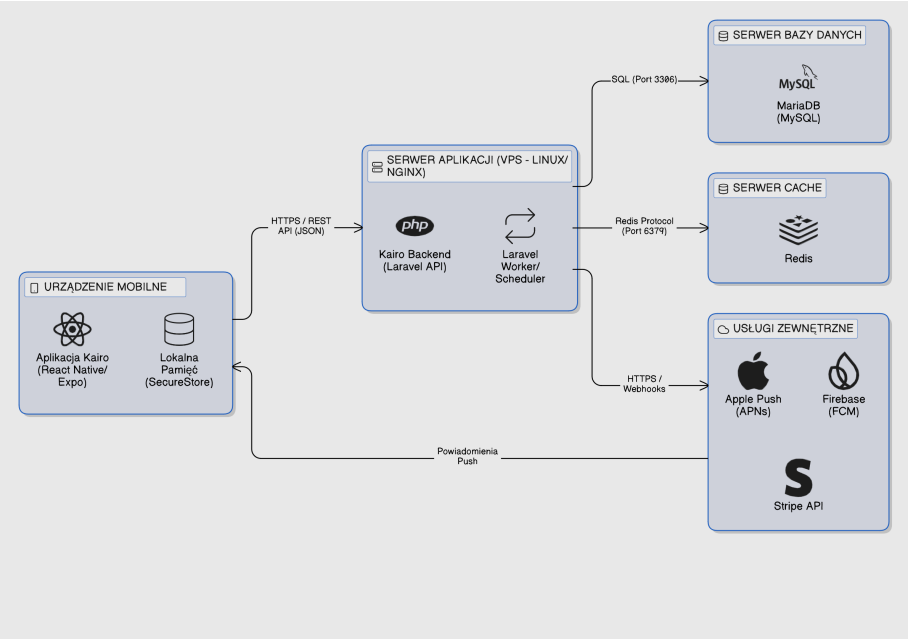
- **Odpowiedzialność:** Zarządzanie wszystkimi operacjami na bazie danych

- **Tabele:**
  - `users` – user ID, email, passwordHash, displayName, createdAt, timezone
  - `habits` – habit ID, user ID, title, description, frequency, color, icon
  - `habit_executions` – execution ID, habit ID, date, status, timestamp
  - `reminders` – reminder ID, habit ID, time, repeatRule, enabled, timezone
  - `friendships` – friendship ID, userId1, userId2, status, createdAt
  - `notifications` – notification ID, recipient ID, type, payload, sentAt, readAt, status
  - `achievements` – achievement ID, habit ID, user ID, name, unlockedAt
  - `user_settings` – settings ID, user ID, theme, notificationsEnabled, language, timezone
- **Indeksy:** Powinny być na: userId (habits, friendships), habitId (habit_executions), recipientId (notifications), createdAt (dla sortowania czasowego)

Przepływ danych między komponentami:

1. User App → **Authentication** → JWT token
2. User App → **Habit Management** (z JWT) → CRUD habits
3. **Habit Management** → **Reminder Scheduler** (z habit details)
4. **Reminder Scheduler** → **Notification Service** (reminder event)
5. **Notification Service** → Firebase/APNs → User Device
6. User App → **Social Features** → Friends management
7. **Social Features** → **Notification Service** (ping events)
8. User App → **Analytics** → Progress data
9. Wszystkie komponenty → **Data Access Layer** → Database

6.7. Diagram Wdrożenia (Deployment Diagram)



Rysunek 7: Diagram wdrożenia Aplikacji Kairo Habit

Opis zastosowania:

Diagram wdrożenia pokazuje rozkład komponentów oprogramowania na konkretne węzły fizyczne (serwery, urządzenia), wraz z komunikacją między nimi.

Węzły wdrożenia:

1. Urządzenie mobilne użytkownika (Mobile Client Layer)

- **Technologia:** React Native (cross-platform iOS/Android)
- **Komponenty na urządzeniu:**
  - UI/UX Layer (React Components, Navigation)
  - Local Storage (SQLite lub Realm dla cache'u offline)
  - API Client (Axios/Fetch API dla REST calls)
  - Push Notification Handler
  - Device storage dla user preferences
- **Komunikacja:** HTTPS (TLS 1.3) → API Backend
- **Dystrybucja:** App Store (iOS), Google Play (Android)

2. Serwer Backend (Application Server)

- **Architektura:** Cloud hosting (AWS EC2, Google Cloud, Azure) lub dedicated server
- **System operacyjny:** Linux (Ubuntu 20.04 LTS)
- **Runtime:** PHP 8.1+ z web serverem Nginx/Apache
- **Komponenty na serwerze:**
  - REST API Server (PHP Framework: Laravel/Symfony)
  - Authentication Service (JWT token management)
  - Business Logic Layers (Habit, Social, Analytics, Notification services)
  - Job Queue (Redis, RabbitMQ dla asynchronicznych zadań)
  - Cache Layer (Redis dla szybkiego dostępu do hot data)
- **Port:** 80 (HTTP), 443 (HTTPS)
- **Komunikacja:**
  - Przychodzące: HTTPS z Mobile App, REST API calls
  - Wychodzące: SQL queries do Database Server, Firebase/APNs API (push notifications)
- **Skalowanie:** Możliwość wdrażania na wielu instancjach za load balancerem (nginx load balancer lub AWS ALB)

3. Serwer bazy danych (Database Server)

- **Technologia:** MySQL 8.0
- **Port:** 3306 (MySQL)
- **Magazyn:** Relacyjna baza danych z następującymi tabelami:

```sql
CREATE TABLE users (
  userId VARCHAR(36) PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  passwordHash VARCHAR(255) NOT NULL,
  displayName VARCHAR(100),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  timezone VARCHAR(50)
);

CREATE TABLE habits (
```

```
  habitId VARCHAR(36) PRIMARY KEY,
  userId VARCHAR(36) NOT NULL,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  frequency ENUM('DAILY', 'WEEKLY', 'CUSTOM'),
  color VARCHAR(7),
  icon VARCHAR(50),
  createdAt TIMESTAMP,
  FOREIGN KEY (userId) REFERENCES users(userId)
);

CREATE TABLE habit_executions (
  executionId VARCHAR(36) PRIMARY KEY,
  habitId VARCHAR(36) NOT NULL,
  date DATE NOT NULL,
  status ENUM('COMPLETED', 'SKIPPED', 'MISSED'),
  timestamp TIMESTAMP,
  FOREIGN KEY (habitId) REFERENCES habits(habitId),
  INDEX idx_habit_date (habitId, date)
);

CREATE TABLE reminders (
  reminderId VARCHAR(36) PRIMARY KEY,
  habitId VARCHAR(36) NOT NULL,
  time TIME NOT NULL,
  repeatRule VARCHAR(50),
  enabled BOOLEAN DEFAULT true,
  timezone VARCHAR(50),
  FOREIGN KEY (habitId) REFERENCES habits(habitId)
);

CREATE TABLE friendships (
  friendshipId VARCHAR(36) PRIMARY KEY,
  userId1 VARCHAR(36) NOT NULL,
  userId2 VARCHAR(36) NOT NULL,
  status ENUM('PENDING', 'ACCEPTED', 'REJECTED'),
  createdAt TIMESTAMP,
  FOREIGN KEY (userId1) REFERENCES users(userId),
  FOREIGN KEY (userId2) REFERENCES users(userId),
  UNIQUE KEY unique_friendship (userId1, userId2)
);

CREATE TABLE notifications (
  notificationId VARCHAR(36) PRIMARY KEY,
  recipientId VARCHAR(36) NOT NULL,
```

```
  type ENUM('REMINDER', 'FRIEND_PING', 'ACHIEVEMENT'),
  payload JSON,
  sentAt TIMESTAMP,
  readAt TIMESTAMP,
  status ENUM('PENDING', 'SENT', 'DELIVERED', 'READ'),
  FOREIGN KEY (recipientId) REFERENCES users(userId),
  INDEX idx_recipient_status (recipientId, status)
);

CREATE TABLE achievements (
  achievementId VARCHAR(36) PRIMARY KEY,
  habitId VARCHAR(36) NOT NULL,
  userId VARCHAR(36) NOT NULL,
  name VARCHAR(100),
  description TEXT,
  unlockedAt TIMESTAMP,
  FOREIGN KEY (habitId) REFERENCES habits(habitId),
  FOREIGN KEY (userId) REFERENCES users(userId)
);

CREATE TABLE user_settings (
  settingsId VARCHAR(36) PRIMARY KEY,
  userId VARCHAR(36) UNIQUE NOT NULL,
  theme ENUM('LIGHT', 'DARK') DEFAULT 'LIGHT',
  notificationsEnabled BOOLEAN DEFAULT true,
  language VARCHAR(5) DEFAULT 'pl',
  timezone VARCHAR(50),
  FOREIGN KEY (userId) REFERENCES users(userId)
);
```

Listing 24: Glowne tabele bazy danych

- **Kopie zapasowe:** Dzienne automatyczne kopie zapasowe (AWS RDS automatyczne kopie lub fizyczne kopie)
- **Replikacja:** Replikacja Master-Slave dla wysokiej dostępności
- **Dostęp:** Ograniczony do Serwera aplikacji (whitelista IP)

4. Cache Server (In-Memory Data Store)

- **Technologia:** Redis 6.0+
- **Zastosowanie:**
  - Session store (JWT tokens)

  - Cache dla często przywoływanych danych (progress statistics, user profiles)
  - kolejka prac dla asynchronicznych procesów (reminder scheduling, notification sending)
  - Strona główna z aktywnością w czasie rzeczywistym

- **Port:** 6379
- **TTL (Time To Live):** Różne - session 24h, cache 1h, queue messages until processed

5. Push Notification Service (External)

- **Firebase Cloud Messaging (FCM)** - dla urządzeń Android
  - Serwery Google'a
  - Wysyłanie push notificatios via REST API
  - Topic subscription dla mass notifications
- **Apple Push Notification Service (APNs)** - dla iOS
  - Serwery Apple'a
  - Certificate-based authentication
  - Device token registration
- **Komunikacja:** Backend Application Server wysyła HTTPS requests do Firebase-/APNs

Przepływ komunikacji:

1. **Użytkownik otwiera aplikacje na urządzeniu mobilnym** → Aplikacja inicjalizuje
2. **Aplikacja** Wysyła rządanie HTTPS do **Backend API Serwera**

```
POST https://api.kairo.app/api/v1/habits/complete
Headers: {Authorization: Bearer <JWT_TOKEN>, Content-Type:
   application/json}
Body: {habitId: "abc-123", date: "2026-01-24"}
```

Listing 25: Example request

3. **Backend API serwer** processuje rządanie:
   - Waliduje JWT (sprawdza Redis cache)

- Wykonuje logikę biznesową
- Pyta **Database Serwer**
- Może składać wyniki w **Redis Cache**

4. **Database Serwer** zwraca dane do Backendu
5. **Backend** zwraca HTTP 200 do aplikacji mobilnej
6. *(Asynchronicznie):* **Kolejka operacji w Backend** (w Redis) triggeruje asynchroniczne wykonanie zadań:
   - Wyślij powiadomienie o Achievement'cie jeśli otrzymano dostateczny Streak
   - Zaktualizuj zcachowane dane użytkownika
   - Zlogguj aktywność użytkownika
7. **Powiadomienia worker** zarządza kolejkami i wiadomościami **Firebase/APNs** do pushowania powiadomień
8. **Urządzenie mobilne** otrzymuje powiadomienie poprzez **Firebase/APNs**
9. **Aplikacja mobilna** Wyświetla powiadomienie użytkownikowi

7. Inne wytyczne dla projektowania wybranego systemu

Oprócz diagramów UML oraz doboru technologii, istotnym elementem procesu projektowego systemu Kairo Habit są dodatkowe wytyczne jakościowe. Wytyczne te nie wynikają bezpośrednio z diagramów, lecz wpływają na użyteczność, bezpieczeństwo oraz niezawodność końcowego rozwiązania.

7.1. Projektowanie interfejsu użytkownika (UI/UX)

Projekt interfejsu użytkownika powinien koncentrować się na prostocie obsługi oraz minimalizacji liczby interakcji wymaganych do wykonania podstawowych czynności. System zarządzania nawykami jest używany w krótkich sesjach, dlatego kluczowe funkcje, takie jak oznaczenie wykonania nawyku czy podgląd postępów, powinny być dostępne w sposób szybki i intuicyjny.

Interfejs powinien zachowywać spójność wizualną oraz przewidywalność zachowań elementów na wszystkich ekranach aplikacji. Istotne jest również uwzględnienie dostępności, w tym odpowiedniego kontrastu kolorów, trybu ciemnego oraz czytelnej typografii. Projekt UI/UX powinien wspierać pozytywne doświadczenie użytkownika i motywować go do regularnego korzystania z aplikacji.

7.2. Bezpieczeństwo danych i zgodność z RODO

System Kairo Habit przetwarza dane osobowe użytkowników, dlatego projektowanie rozwiązania musi uwzględniać zasady bezpieczeństwa informacji oraz wymagania Rozporządzenia o Ochronie Danych Osobowych (RODO). Należy stosować zasadę minimalizacji danych, przechowując wyłącznie informacje niezbędne do realizacji funkcjonalności systemu.

Projekt powinien zapewniać separację danych wrażliwych od danych operacyjnych, a także umożliwiać użytkownikowi realizację jego praw, takich jak wgląd do danych, ich modyfikacja oraz usunięcie konta. Bezpieczeństwo powinno być uwzględnione już na etapie projektowania systemu (privacy by design), a wszystkie operacje na danych użytkownika powinny być możliwe do audytowania.

7.3. Strategia testowania

Zapewnienie wysokiej jakości systemu wymaga zastosowania wielopoziomowej strategii testowania. Testy jednostkowe powinny weryfikować poprawność działania logiki biznesowej, w szczególności mechanizmów związanych z nawykami, przypomnieniami oraz systemem osiągnięć.

Testy integracyjne pozwalają sprawdzić poprawność współpracy pomiędzy aplikacją mobilną, backendem oraz bazą danych. Uzupełnieniem są testy end-to-end, które symulują rzeczywiste scenariusze użytkownika. Dodatkowo system powinien być poddawany testom wydajnościowym oraz regresyjnym, aby zapewnić stabilność działania w miarę rozwoju funkcjonalności.

8. Wnioski

Przeprowadzona analiza oraz projekt systemu Kairo Habit pokazują, że zastosowanie notacji UML znacząco wspiera proces projektowania systemów informatycznych. Diagramy UML umożliwiają uporządkowanie wymagań, lepsze zrozumienie struktury systemu oraz wczesną identyfikację potencjalnych problemów projektowych.

Zaproponowana architektura systemu zapewnia spójność pomiędzy wymaganiami funkcjonalnymi i niefunkcjonalnymi a warstwą implementacyjną. Uwzględnienie aspektów takich jak skalowalność, bezpieczeństwo oraz wydajność sprawia, że system jest przygotowany do pracy w środowisku produkcyjnym oraz dalszego rozwoju.

Projekt Kairo Habit stanowi przykład kompleksowego podejścia do tworzenia systemu informatycznego, łączącego analizę wymagań, modelowanie UML, decyzje architektoniczne oraz wytyczne jakościowe. Takie podejście zwiększa szanse na stworzenie rozwiązania stabilnego, użytecznego i łatwego w utrzymaniu.