JSONPlaceholder pobieranie postów

Dokumentacja techniczna

Maciej Kasik

5 maja 2025

Spis treści

1 Opis projektu

1.1 Cel i zakres

Aplikacja służy do pobierania postów z serwisu JSONPlaceholder API i zapisywania ich w postaci indywidualnych plików JSON. Każdy post jest zapisywany w oddzielnym pliku o nazwie odpowiadającej identyfikatorowi posta.

Aplikacja obsługuje trzy środowiska pracy:

- **Development** – środowisko deweloperskie do pracy lokalnej
- **Staging** – środowisko testowe
- **Production** – środowisko produkcyjne

1.2 Technologie

- Kotlin 1.9.0 - język programowania
- Retrofit 2.9.0 - komunikacja HTTP
- OkHttp 4.10.0 - klient HTTP
- Gson 2.10.1 - obsługa formatu JSON
- Coroutines 1.7.3 - programowanie asynchroniczne
- Gradle - system budowania

1.3 Wymagania systemowe

- JDK 8+
- Dostęp do internetu

2 Wymagania projektowe

2.1 Wymagania funkcjonalne

Poniższe wymagania funkcjonalne określają, co system musi robić, aby zaspokoić potrzeby użytkownika:

1. **Pobieranie danych** – System musi pobierać kompletną listę postów z API JSON-Placeholder poprzez wywołanie endpointu `/posts`.
2. **Zapisywanie plików** – Każdy pobrany post musi zostać zapisany jako osobny plik w formacie JSON.
3. **Identyfikacja plików** – Nazwy plików wyjściowych muszą odpowiadać identyfikatorom postów (np. `1.json`, `2.json`).
4. **Katalog docelowy** – System musi automatycznie tworzyć katalog docelowy, jeśli ten nie istnieje w momencie uruchomienia aplikacji.

5. **Raportowanie** – System musi raportować postęp operacji w konsoli, w tym:
    - liczbę pobranych postów
    - liczbę pomyślnie zapisanych plików
    - podsumowanie całej operacji
6. **Obsługa błędów** – System musi wykrywać i informować o błędach występujących podczas:
    - komunikacji z API
    - przetwarzania danych
    - operacji zapisu na dysku

2.2 Wymagania niefunkcjonalne

Poniższe wymagania niefunkcjonalne określają, jak system ma działać i jakie aspekty jakościowe powinien spełniać:

1. **Wydajność** – System musi wykorzystywać asynchroniczne mechanizmy komunikacji sieciowej, aby zminimalizować czas oczekiwania na odpowiedź API.
2. **Niezawodność** – System musi obsługiwać sytuacje wyjątkowe, takie jak:
    - brak dostępu do sieci
    - niedostępność usługi API
    - brak uprawnień do zapisu na dysku
3. **Modularność** – Kod systemu musi być zorganizowany w logiczne komponenty o jednorodnych odpowiedzialnościach, zgodnie z zasadami Clean Architecture.
4. **Rozszerzalność** – System musi umożliwiać łatwe rozszerzenie o dodatkowe funkcjonalności, takie jak:
    - obsługa innych endpointów JSONPlaceholder API
    - konfigurację parametrów pracy (np. wybór katalogu docelowego)
    - filtry danych
5. **Bezpieczeństwo** – System musi wykonywać bezpieczne operacje wejścia/wyjścia, unikając typowych błędów, takich jak race conditions czy wycieki zasobów.
6. **Utrzymywalność** – Kod źródłowy musi być dokumentowany zgodnie ze standardami, a architektura musi ułatwiać jego późniejsze modyfikacje.

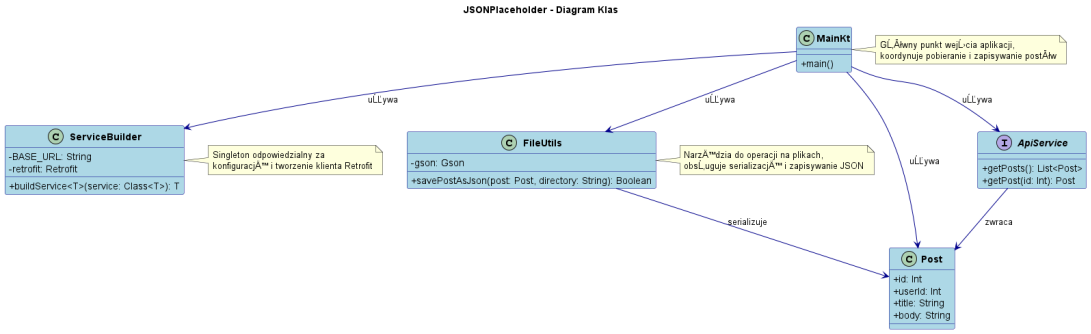
3 Architektura systemu

3.1 Diagramy UML

W celu lepszego zobrazowania architektury systemu, poniżej przedstawiono diagramy UML: diagram klas, diagram sekwencji oraz diagram komponentów.

3.1.1 Diagram klas

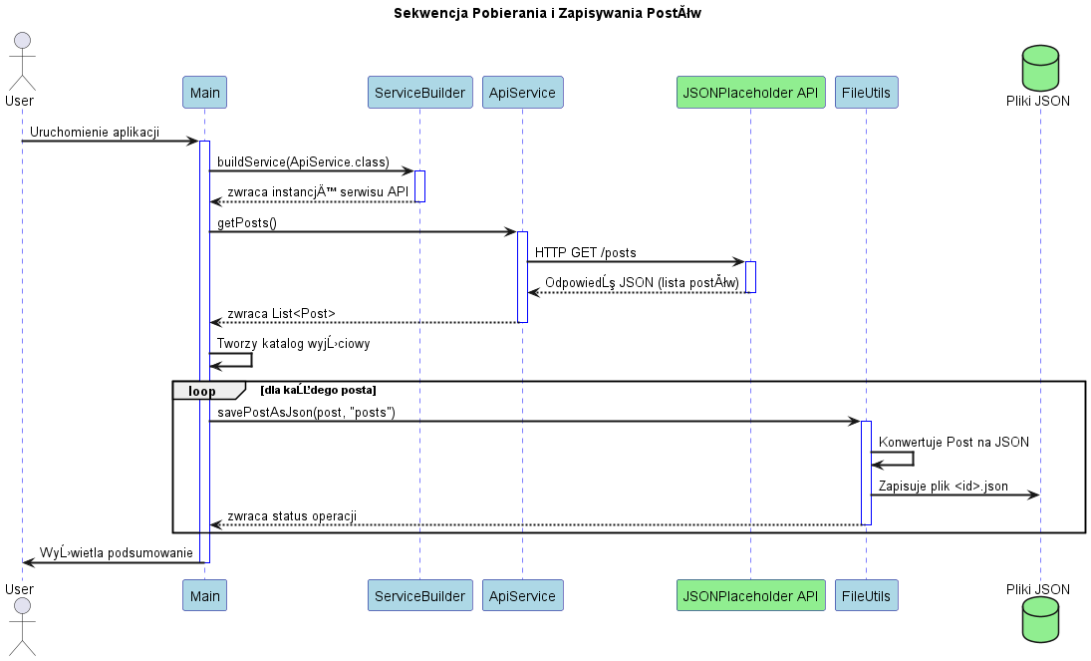
Diagram klas przedstawia strukturę statyczną systemu, pokazując klasy, ich atrybuty, metody oraz relacje między nimi.



Rysunek 1: Diagram klas systemu pobierania postów

3.1.2 Diagram sekwencji

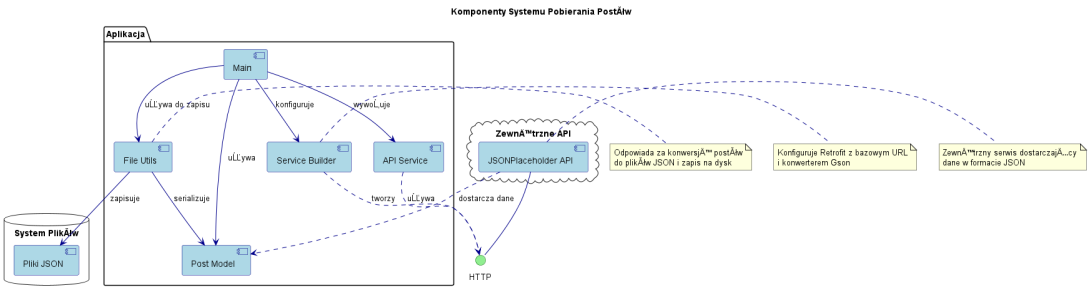
Diagram sekwencji ilustruje interakcje między komponentami systemu w czasie, pokazując przepływ komunikacji podczas pobierania i zapisywania postów.



Rysunek 2: Diagram sekwencji pobierania i zapisywania postów

3.1.3 Diagram komponentów

Diagram komponentów przedstawia komponenty systemu oraz ich wzajemne zależności, ukazując ogólną strukturę aplikacji.



Rysunek 3: Diagram komponentów systemu

3.2 Opis komponentów

- **Model Post** - klasa danych reprezentująca strukturę posta: id, userId, title, body
- **Serwis API** - interfejs definiujący endpointy API:
  - `getPosts()` - pobieranie listy wszystkich postów
  - `getPost(id)` - pobieranie pojedynczego posta
- **Konstruktor Serwisu** - fabryka klientów REST:
  - Konfiguracja bazowego URL dla aktualnego środowiska
  - Konfiguracja konwertera JSON (Gson)
  - Konfiguracja klienta HTTP z parametrami zależnymi od środowiska
  - Włączanie/wyłączanie logowania na podstawie konfiguracji środowiskowej
- **Narzędzia Plikowe** - operacje na plikach:
  - Serializacja obiektów do formatu JSON
  - Tworzenie katalogów
  - Zapisywanie danych do plików
  - Czyszczenie katalogów wyjściowych (w trybie development)
- **Główna Aplikacja** - przepływ logiki biznesowej:
  - Przetwarzanie parametrów wywołania
  - Wybór środowiska uruchomieniowego
  - Pobieranie postów z API
  - Koordynacja przetwarzania danych
  - Zapisywanie do plików
- **Konfiguracja Środowiskowa** - zarządzanie konfiguracją:
  - Określenie parametrów dla różnych środowisk
  - Wybór konfiguracji na podstawie parametrów lub zmiennych środowiskowych
  - Dostarczanie ustawień dla pozostałych komponentów

4 Przepływ danych

4.1 Pobieranie danych

1. Utworzenie instancji serwisu API za pomocą konstruktora serwisu
2. Wysłanie asynchronicznego żądania HTTP GET do `/posts`
3. Deserializacja odpowiedzi JSON do listy obiektów Post

4.2 Przetwarzanie i zapisywanie

1. Utworzenie katalogu wyjściowego `posts` (jeśli nie istnieje)
2. Iteracja przez listę pobranych postów
3. Dla każdego posta:
   - Konwersja obiektu Post do sformatowanego ciągu JSON
   - Zapisanie danych do pliku `<id>.json`
4. Raportowanie postępu w konsoli

5 Struktura projektu

5.1 Organizacja kodu

- `Post.kt` - model danych
- `ApiService.kt` - definicja interfejsu API
- `ServiceBuilder.kt` - fabryka klientów HTTP
- `FileUtils.kt` - narzędzia plikowe
- `Main.kt` - punkt wejściowy aplikacji

5.2 Konfiguracja Gradle

Projekt wykorzystuje system budowania Gradle z następującymi zależnościami:

- Retrofit 2.9.0 - klient HTTP
- Gson 2.10.1 - biblioteka JSON
- Kotlinx Coroutines 1.7.3 - wsparcie dla asynchroniczności

6 Instrukcja użytkowania

6.1 Budowanie i uruchamianie

Aplikacja może zostać zbudowana i uruchomiona za pomocą następujących komend:

```
# Budowanie aplikacji
./gradlew build

# Uruchamianie aplikacji (domy lnie w rodowisku development)
./gradlew run
```

6.2 Uruchamianie w różnych środowiskach

Aplikacja obsługuje trzy środowiska uruchomieniowe. Można je wybrać na kilka sposobów:

6.2.1 Za pomocą dedykowanych zadań Gradle

```
# rodowisko deweloperskie
./gradlew runDev

# rodowisko testowe
./gradlew runStaging

# rodowisko produkcyjne
./gradlew runProd
```

6.2.2 Za pomocą parametrów wiersza poleceń

```
# rodowisko deweloperskie
java -jar app.jar --env=dev

# rodowisko testowe
java -jar app.jar --env=staging

# rodowisko produkcyjne
java -jar app.jar --env=prod
```

6.2.3 Za pomocą zmiennej środowiskowej

```
# rodowisko deweloperskie
APP_ENV=dev java -jar app.jar

# rodowisko testowe
APP_ENV=staging java -jar app.jar

# rodowisko produkcyjne
APP_ENV=prod java -jar app.jar
```

6.3 Różnice między środowiskami

Cecha	Development	Staging	Production
Katalog wyjściowy	`posts_dev`	`posts_staging`	`posts`
Timeout żądań	30 sekund	20 sekund	10 sekund
Logowanie HTTP	Włączone	Włączone	Wyłączone
Czyszczenie katalogu	Przy starcie	Brak	Brak

6.4 Wynik działania

Po uruchomieniu aplikacja:

1. Wyświetla informacje o aktualnym środowisku pracy
2. Pobiera posty z API JSONPlaceholder
3. Tworzy odpowiedni katalog wyjściowy (zależny od środowiska)
4. W trybie deweloperskim czyści katalog wyjściowy przed zapisem
5. Zapisuje posty jako pliki JSON o nazwach odpowiadających identyfikatorom
6. Wyświetla podsumowanie operacji (liczba pobranych i zapisanych postów)

6.5 Format danych wyjściowych

Każdy pobrany post jest zapisywany w pliku JSON o nazwie `<id>.json`. Struktura pliku zawiera następujące pola:

- `id` - unikalny identyfikator posta (liczba całkowita)
- `userId` - identyfikator użytkownika (liczba całkowita)
- `title` - tytuł posta (tekst)
- `body` - treść posta (tekst)

7 Możliwości rozwoju

7.1 Proponowane rozszerzenia

- Implementacja równoległego przetwarzania dla szybszego pobierania i zapisywania
- Dodanie interfejsu graficznego
- Rozszerzenie funkcjonalności o pozostałe endpointy JSONPlaceholder API:
  - `/comments` - komentarze do postów
  - `/albums` - albumy zdjęć
  - `/photos` - zdjęcia
  - `/todos` - zadania do wykonania

  - `/users` - dane użytkowników

- Konfigurowalne parametry specyficzne dla każdego środowiska (np. limity pobieranych postów)
- Implementacja mechanizmu migracji danych między środowiskami
- Implementacja systemu raportowania i logowania

7.2 Optymalizacje

- Buforowanie żądań sieciowych
- Zarządzanie przetwarzaniem równoległym
- Optymalizacja operacji I/O
- Implementacja mechanizmu ponownych prób dla nieudanych żądań
- Automatyczne przełączanie między środowiskami na podstawie metryk wydajnościowych
- Dodanie mechanizmu monitorowania i zbierania metryk dla każdego środowiska

8 Podsumowanie

Aplikacja "JSONPlaceholder pobieranie postów"demonstruje poprawną implementację klienta API REST w języku Kotlin z wykorzystaniem nowoczesnych bibliotek i wzorców programistycznych. System jest modułowy, rozszerzalny i obsługuje trzy środowiska uruchomieniowe (development, staging, production), co zapewnia elastyczność i dostosowanie do różnych etapów cyklu życia oprogramowania.