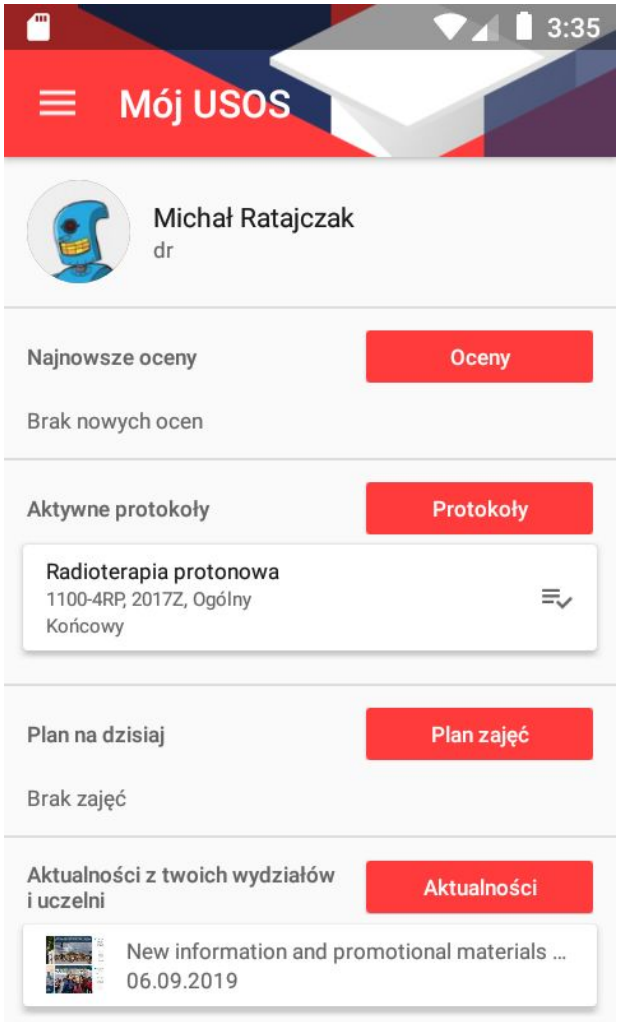
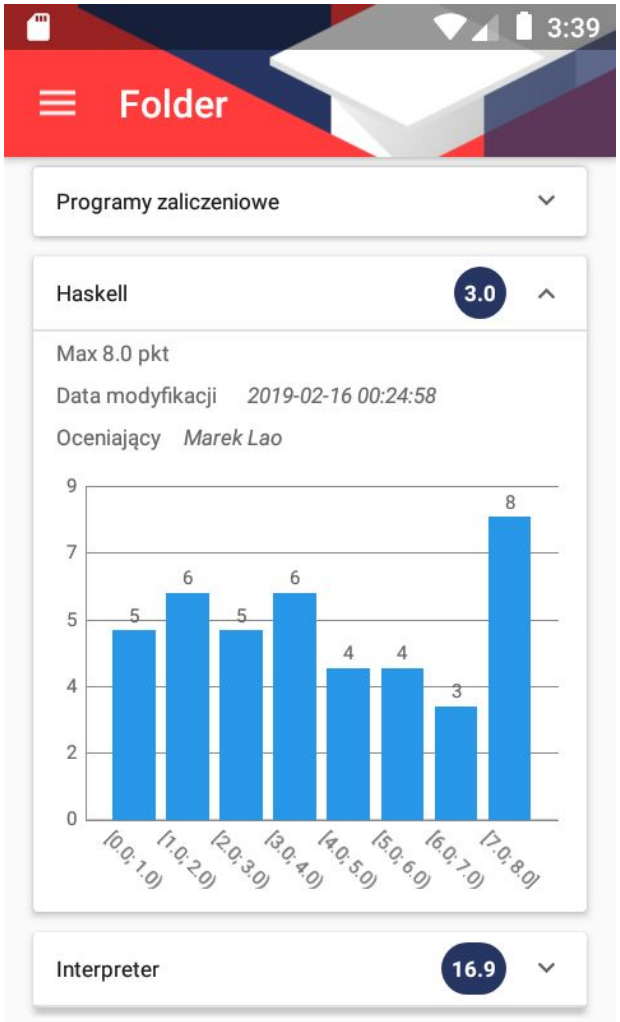
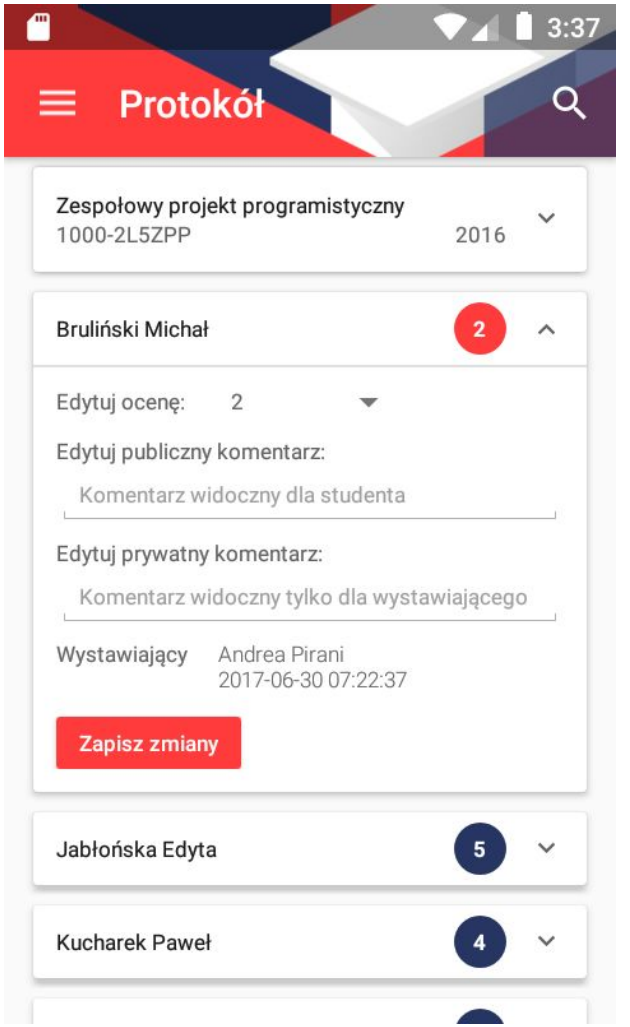
Mobilny USOS - aplikacja mobilna dla studentów i pracowników uczelni

Dominik Murzynowski (współautor pracy), **Piotr Zalas**

Mobilny USOS





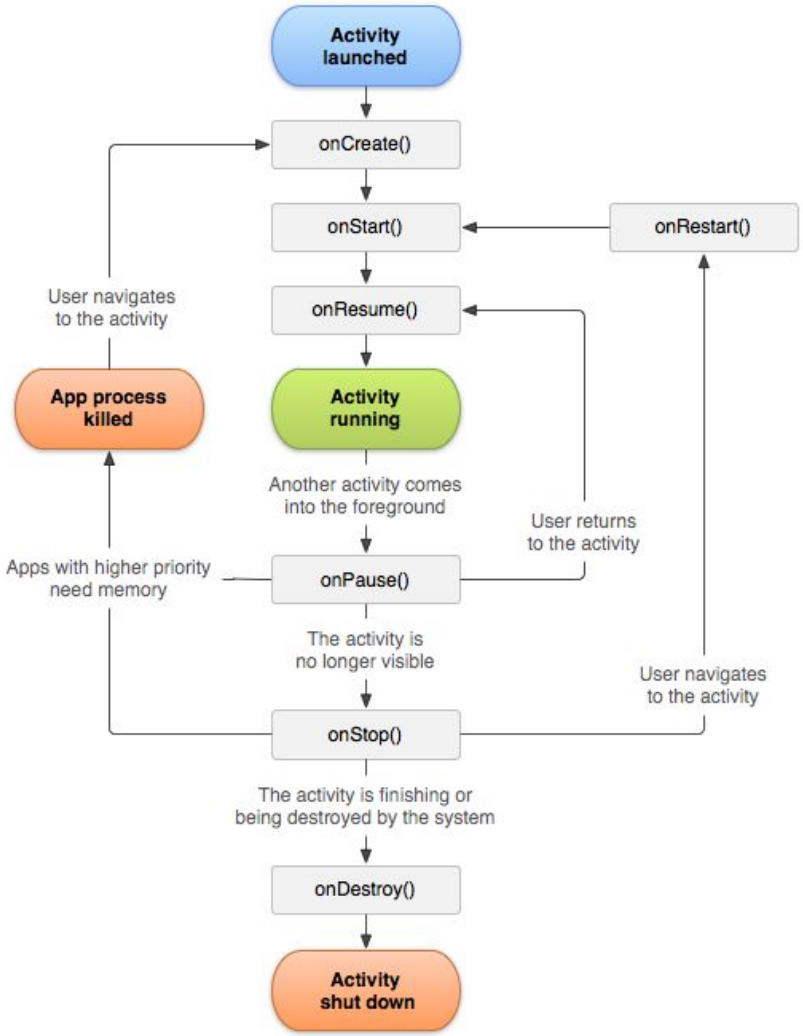


Kilka informacji technicznych - Android

- Manifest - z niego system odczytuje informacje o aplikacji i nazwę activity, która ma być wykonana po uruchomieniu aplikacji.
- Activity - Jeden ekran aplikacji (ekran powitalny, ekran logowania, ekran główny). Może się składać z fragmentów.
- Fragment - Wydzielony fragment ekranu w activity. Activity zarządza menu i belką tytułową, a fragment implementuje resztę widoku, np. listę protokołów, widok folderu z sprawdzianami.
- Widok - np. przycisk, element listy albo inny reużywalny element UI.

- Model zdarzeniowy - programista nie widzi explicite pętli zdarzeń.
- Fragment vs widok - fragment ma swój cykl życia powiązany z activity (i jest go świadomy), widok nie.

Kilka informacji technicznych - Android



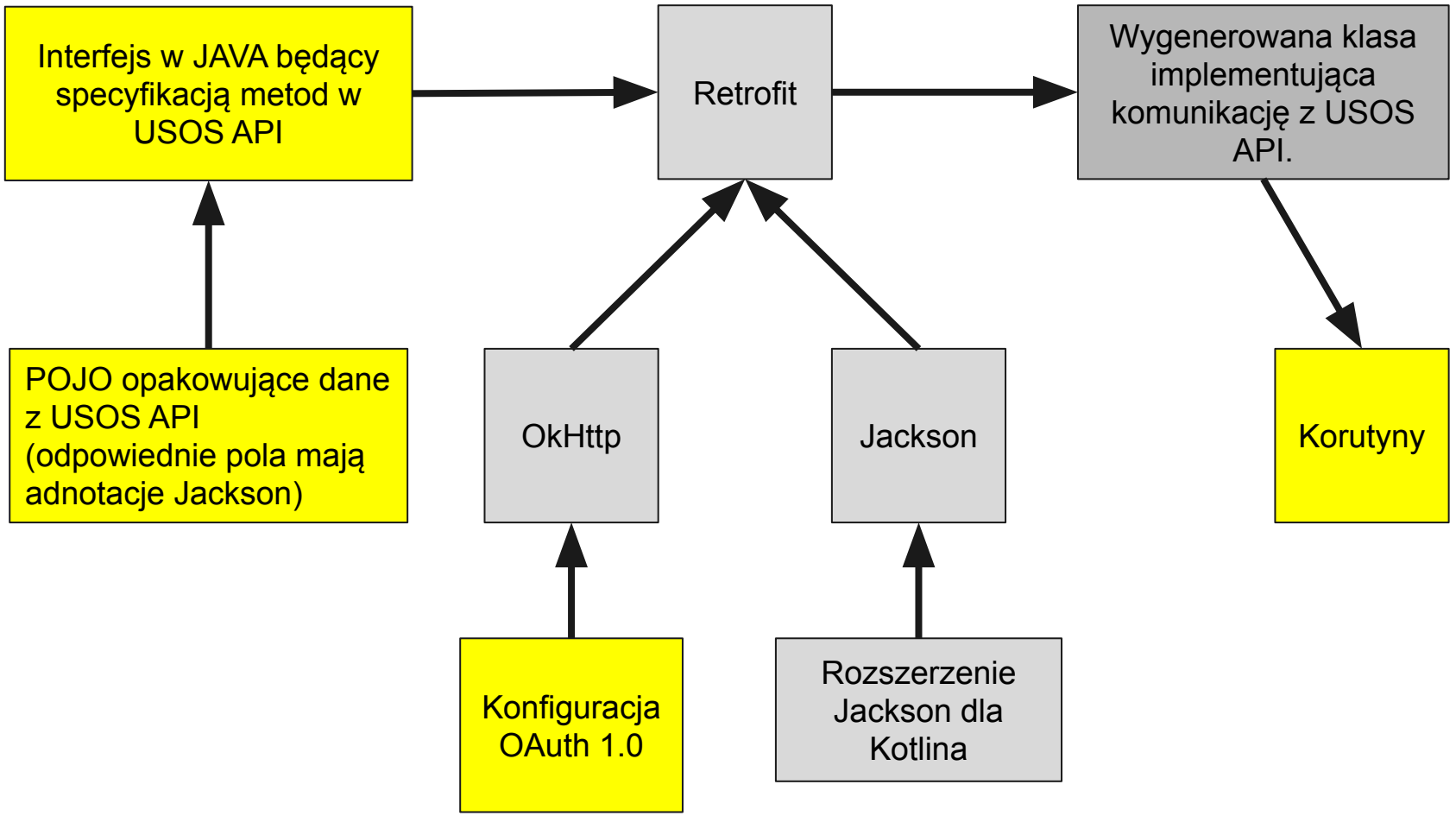
System budowania - Gradle

- Każda uczelnia ma swój wariant aplikacji.
- Poszczególne warianty różnią się konfiguracją, więc każda uczelnia ma swój katalog ze swoimi danymi konfiguracyjnymi.
- Oprócz tego jest osobny katalog z głównym kodem źródłowym.
- Podczas kompilacji główny kod i katalog uczelni są scalane (modyfikacje uczelni mają pierwszeństwo).
- Scalane są m. in. pliki z kodem (o ile nie konfliktują ze sobą), grafiki, tłumaczenia napisów, stałe konfiguracyjne.
- Na zakończenie gotowa paczka jest obfuskowana: usuwane są niepotrzebne adnotacje w plikach klas, następuje propagacja stałych, nazwy pakietów, klas i funkcji są zmieniane na losowe. Trzeba uważać, bo powoduje to problemy przy użyciu refleksji.

Komunikacja z USOS API

- Zapytania wykonywane jako metoda POST protokołu HTTP(S), odpowiedź w formacie JSON.
- Autoryzacja przez protokół OAuth 1.0a.
- Powiadomienia o zdarzeniach są dostarczane za pośrednictwem Firebase Cloud Messaging.
- Komunikacja po stronie aplikacji jest asynchroniczna.

Komunikacja z USOS API



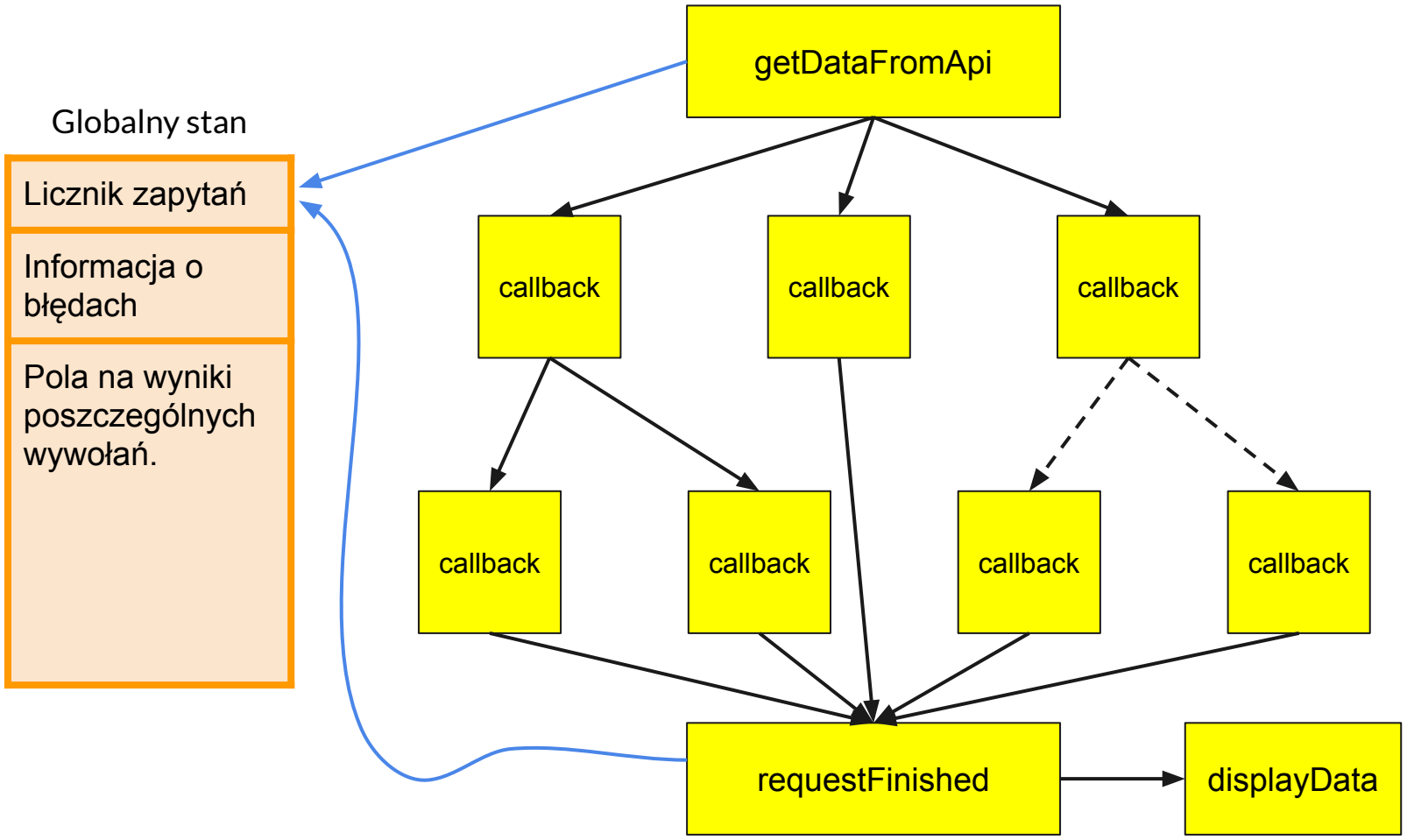
Problemy Mobilnego USOS

- Dużo powtarzalnego kodu, zwłaszcza związanego z komunikacją z USOS API.
- Mimo dużej powtarzalności kodu, niespójności między fragmentami np. przy obsłudze cyklu życia fragmentu.
- Brak separacji odpowiedzialności w fragmencie, każdym ekranem aplikacji niemal w całości zajmowały się wielkie klasy mające 1000+ linijek kodu.
- Notoryczne awarie wynikające z błędów programistycznych (zazwyczaj NullPointerException) i wyścigów przy pobieraniu danych z USOS API.

Kotlin

- Język bardziej zwięzły niż Java, bardzo dużo wbudowanego lukru syntaktycznego: data klasy, automatyczne gettery i settery, dekoratory klas.
- Obsługa nulli wbudowana w system typów oraz przydatne operatory do operowania na referencjach, które mogą być nullami: ?:, ?..
- Słaba inferencja typów - smart casty.
- Naśladuje języki funkcyjne: rozróżnienie na mutowalne i niemutowalne zmienne, typy w kolekcjach domyślnie niemutowalne, implementujące wiele operacji funkcyjnych takich jak map, filter, groupBy, associate, all, any…
- Bogata biblioteka standardowa pomaga omijać ograniczenia okrojonej biblioteki standardowej Java wbudowanej w system Android.
- Eksperymentalna obsługa korutyn wbudowana w język.

Refaktoring komunikacji z USOS API

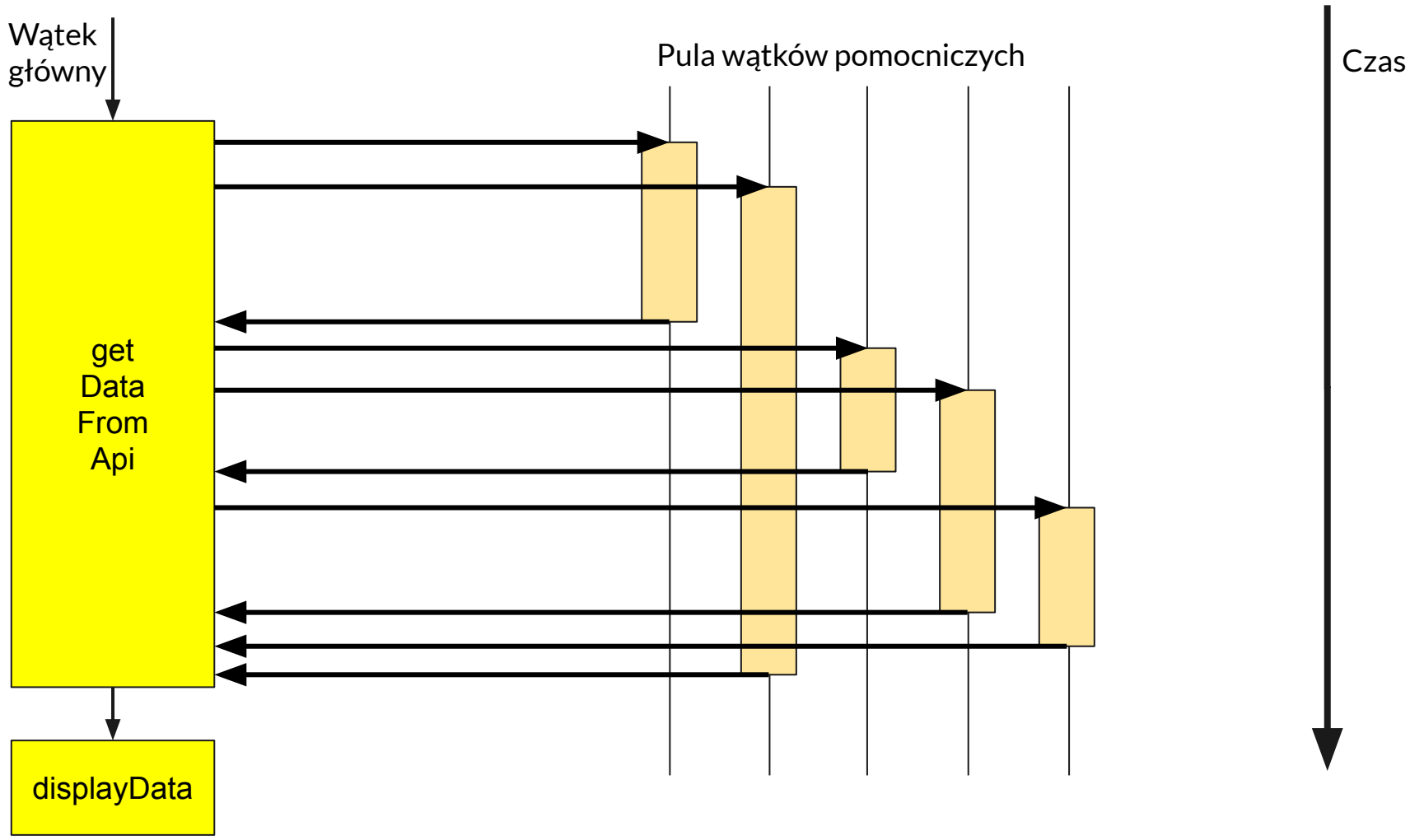


Refaktoring komunikacji z USOS API

Wady starego mechanizmu:

- Konieczność wyliczania, ile zapytań zostało do wykonania - nieciekawe, jeżeli wykonanie niektórych zapytań zależy od wyniku wcześniejszego zapytania w łańcuszku albo jedno z zapytań zakończy się błędem.
- Globalny obiekt ze stanem, do którego dostęp nie był chroniony (można było uruchomić dwa równoległe łańcuchy zapytań, które wzajemnie niszczyły sobie stan).
- Skomplikowanie, graf wywołań nie był widoczny na pierwszy rzut oka.
- Trudne dodawanie nowego wywołania USOS API do łańcucha.
- Powtarzalny kod obsługujący każde wywołanie nie był wyabstrahowany.

Refaktoring komunikacji z USOS API



Refaktoring komunikacji z USOS API

Zalety nowego mechanizmu:

- Programista nie musi już wyliczać liczby wywołań USOS API.
- Wyeliminowanie globalnego stanu, przez co nawet jeśli uruchomimy kilka łańcuchów zapytań, to one nie będą ze sobą interferować.
- Prosta, liniowa reprezentacja grafu wywołań.
- Łatwe rozszerzanie łańcucha o nowe wywołania.
- Ukrycie całego powtarzalnego kodu i obsługi błędów, które są teraz zgłaszane jako wyjątki.

Refaktoring fragmentów

Praktycznie wszystkie fragmenty reimplementowały (z drobnymi zmianami) następujący cykl życia:

- Ustaw tytuł, zaznaczoną pozycję menu i skonfiguruj lupkę wyszukiwarki w osadzającej activity.
- Stwórz layout (widok) fragmentu na ekranie.
- Załaduj dane z cache jeżeli nie są przeterminowane.
- Pobierz dane z USOS API.
- Zapisz pobrane dane w cache.
- Wyświetl pobrane dane na ekranie.
- Jeżeli użytkownik sobie tego zażyczy, zainicjuj pobieranie danych jeszcze raz.

Niektóre fragmenty źle implementowały przypadki brzegowe tego cyklu (np. obsługę sytuacji, gdy internet jest niedostępny).

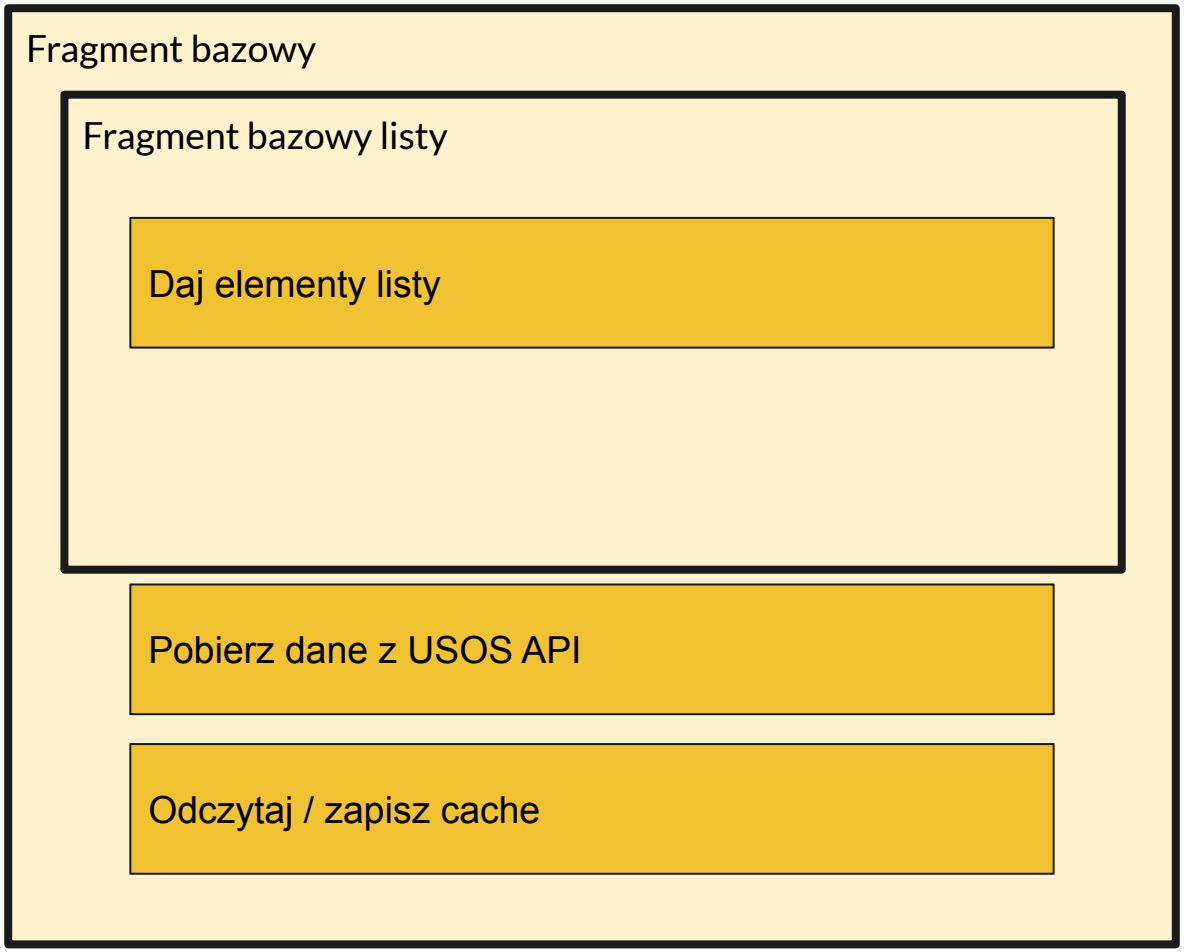
Refaktoring fragmentów

Fragment bazowy

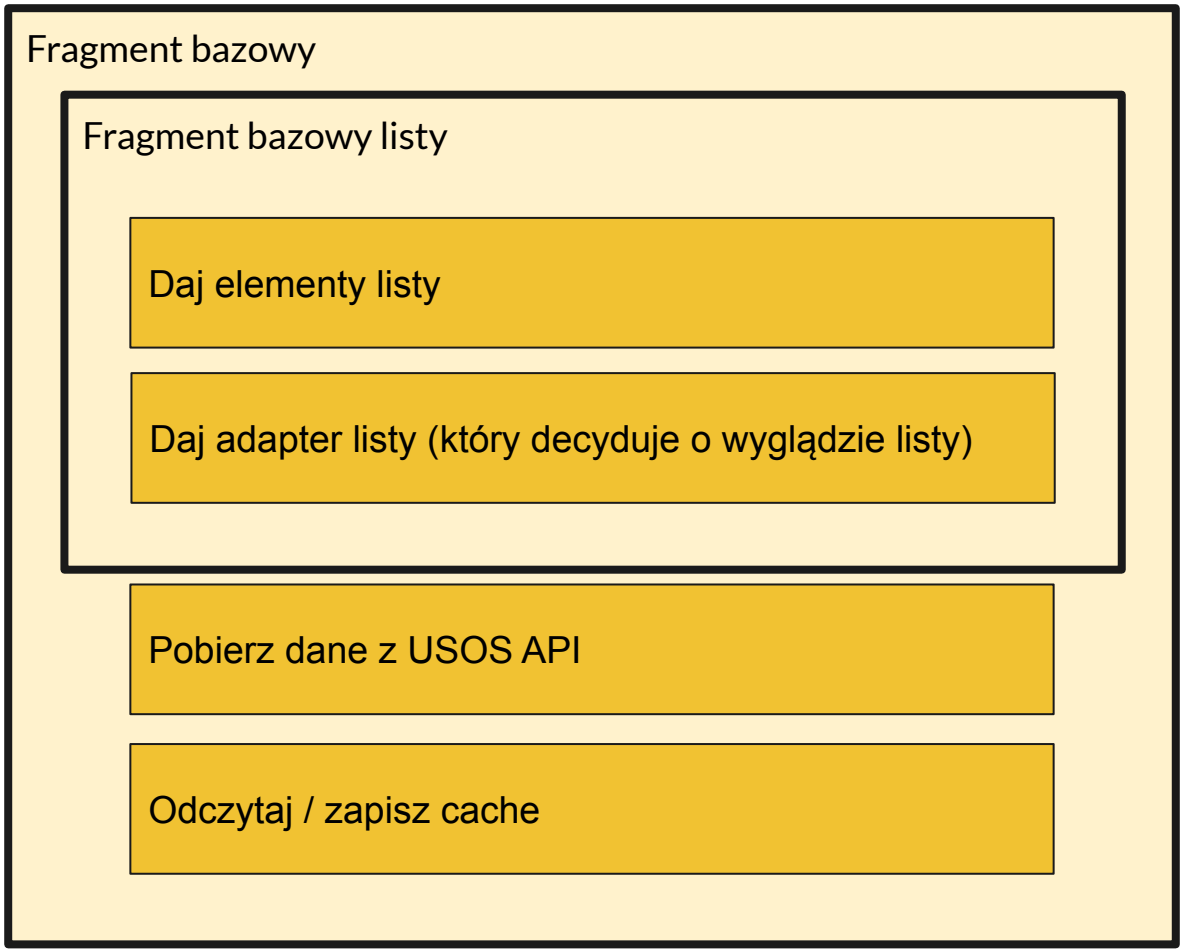
Decyduje, jaka funkcja kiedy się wykona, zarządza ważnością danych z cache, obsługuje i wyświetla błędy z USOS API, zapobiega wyścigom.

- Stwórz layout
- Pobierz dane z USOS API
- Odczytaj / zapisz cache
- Wyświetl dane

Refaktoring fragmentów



Refaktoring fragmentów



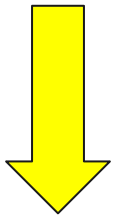
Monitorowanie błędów aplikacji

- Wykorzystaliśmy narzędzie Crashlytics / Fabric.
- Podczas uruchamiania pierwszego activity Mobilny USOS ładuje bibliotekę Crashlytics.
- Crashlytics przechwytuje wszystkie niezłapane wyjątki z całej aplikacji, ze wszystkich wątków. Po złapaniu wyjątku wysyła go na serwer Fabric.
- Fabric oferuje podgląd wszystkich błędów, grupuje podobne wyjątki, zlicza proste statystyki pomagające znaleźć przyczynę błędu.
- Wszystko to jest niezbyt przydatne przy zbyt agresywnej obfuskacji. W takiej sytuacji należy wspomagać się plikami rozjaśniania kodu generowanymi podczas obfuskacji.
- Podobna funkcjonalność jest zapewniana przez sklep Google Play out-of-the-box.

Poprawa odporności na błędy

```
Obiekt input = …;
String output;

if (input != null && input.a() != null && input.a().b() != null && ...) {
        output = input.a().b()...;
} else {
        output = “ABCD”;
}
```



```
val input: Obiekt? = …;
val output: String = input?.a()?.b()?.... ?: “ABCD”;
```

Dziękuję za uwagę