BO·TT·EGA
IT minds

Program szkolenia:

Clean Android

Informacje:

Nazwa:	**Clean Android**
Kod:	**android-clean**
Kategoria:	Android
Grupa docelowa:	developerzy
Czas trwania:	3 dni
Forma:	40% wykłady / 60% warsztaty

Szkolenie zostało przygotowane z myślą o programistach Android, którzy szukają lepszych sposobów na strukturyzację architektury aplikacji i organizację porządku w kodzie.

Podczas szkolenia tworzona jest od podstaw aplikacja oparta o Clean Architecture oraz rozwiązania, które z niej wynikają: testability logiki biznesowej i modularyzacja.

Materiały wstępne

Przed szkoleniem możesz zapoznać się z serią naszych artykułów: Zaawansowane programowanie na platformie Android.

Zalety szkolenia:

- Aspekty bezpieczeństwa
- Architektura i wzorce projektowe
- Bazujemy na Clean Architecture

BOTTEGA IT minds

Szczegółowy program:

1. Testowanie

1.1. Podstawy testowania

- 1.1.1. Rodzaje testów i przykłady ich wykorzystania
- 1.1.2. Zakres testów (Jednostkowe, Integracyjne, End 2 End)
- 1.1.3. Rola testów (Akceptacyjne, Regresywne)
- 1.1.4. Strategia budowania piramidy testów

1.2. Projektowanie przypadków testowych

- 1.2.1. Testowanie przypadków granicznych
- 1.2.2. Nacisk na testy jednostkowe w celu osiągnięcia wysokiego pokrycia testami
- 1.2.3. Rozwarstwienie logiki na Aplikacyjną i Domenową
- 1.2.4. Modelowanie logiki przy pomocy Building Blocks z Domain Driven Design
- 1.2.5. Powtarzalność testów, wyeliminowanie losowości z testów
- 1.2.6. Najlepsze praktyki tworzenia przypadków testowych

1.3. Testowanie jednostkowe

- 1.3.1. Określanie zakresu jednostki i nazwy testów
- 1.3.2. Implementacja testów (JUnit i Spock)
- 1.3.3. Tworzenie własnych asercji
- 1.3.4. Techniki: Mock, Stub, Fake (Mockito i Spock)
- 1.3.5. Dobór technik do potrzeb - czym się kierować
- 1.3.6. Zalety i wady testowania w izolacji
- 1.3.7. Nagrywanie zachowania
- 1.3.8. Weryfikacja wywołań
- 1.3.9. Wprowadzenie do testowania z użyciem narzędzia Spock

BO·TT·EGA
IT minds

1.4. Android

1.4.1. Testowanie w środowisku Androida

1.4.2. Mockowanie w środowisku Androida

1.4.3. Narzędzia, pułapki, najlepsze praktyki

1.4.4. Tworzenie szybkich testów z wykorzystaniem biblioteki Robolectric

1.4.5. Werbalizacja asercji z użyciem AssertJ oraz rozszerzenia AssertJ - Android

1.4.6. Przegląd narzędzi wspomagających testowanie

2. Tworzenie modularnego i testowalnego kodu

2.1. Testability - podatność kodu na testy

2.1.1. Jak pisać kod, który daje się testować

2.1.2. Najlepsze praktyki: SOLID

2.1.3. Wybrane wzorce projektowe, które zwiększają testability: Factory, Strategy, Value Object

2.1.4. Code smell - "zapachy" kodu niepodatnego na testowanie

2.2. Techniki Inversion of Control

2.2.1. Dependency Injection (Dynamiczne wstrzykiwanie zależności)

2.2.1.1. Wstrzykiwanie zaślepek na czas testów

2.2.2. Events

2.2.2.1. Separacja i decoupling testowanych modułów

2.2.2.2. Luźne wiązanie komponentów z użyciem Event Bus

2.2.3. Dobre praktyki i użyteczne wzorce projektowe na platformie Android

2.3. Clean Architecture

2.3.1. Separacja logiki aplikacji od frameworka

2.3.2. Zastosowanie w praktyce zasad rządzących architekturą

2.3.3. Dependency Inversion Principle w kontekście CA

2.3.4. Testowanie poszczególnych elementów architektury

BOTTEGA IT minds

2.3.5. Model View Presenter jako narzędzie do separacji widoku i logiki aplikacji

2.3.5.1. integracja z Androidowym cyklem życia komponentów widoku

2.3.5.2. współbieżne wykonywanie operacji w prezenterze i komunikacja z widokiem

3. Czysty kod oraz poprawa wydajności w warstwie prezentacji aplikacji Androidowych

3.1. Zaawansowane tworzenie interfejsu użytkownika.

3.1.1. Tworzenie własnych elementów interfejsu użytkownika

3.1.1.1. Agregacja wielu widoków w celu wielokrotnego użycia i zastosowania SRP w warstwie widoku

3.1.1.2. Bezpośrednie rysowanie własnych elementów interfejsu użytkownika

3.1.1.3. Stylowanie własnych widoków

3.1.1.4. Obsługa interakcji użytkownika

3.1.1.5. Zarządzanie stanem w widokach

3.2. Optymalizacja działania aplikacji Android

3.2.1. Profilowanie działania aplikacji

3.2.2. Optymalizacje warstwy widoku

3.2.3. Optymalizacje widoków opartych o adaptery (ListView / RecyclerView)

3.2.4. Wykrywanie i zapobieganie wyciekom pamięci

3.2.5. Wydajne zapisywanie stanu

3.2.6. Optymalizacja kodu pod kątem maszyny wirtualnej Dalvik

4. Obfuskowanie i optymalizowanie kodu przy użyciu narzędzia proguard