



Programowanie urządzeń mobilnych. Android

Karol Przystalski



Poznajmy się

2017 - doktorat uzyskany w PAN oraz UJ

od 2010 - CTO @ **Codete**

2007 - 2009 - Software Engineer @ **IBM**

Praca naukowa

Multispectral skin patterns analysis using fractal methods}, K. Przystalski and M. J. Ogorzalek. Expert Systems with Applications, 2017

<https://www.sciencedirect.com/science/article/pii/S0957417417304803>

kprzystalski@gmail.com

karol@codete.com



Zakres materiału

1. Podstawy języka Kotlin
2. Zaawansowany Kotlin
3. Podstawy systemu Android
4. Aktywności oraz podstawowe wzorce projektowe na platformie Android
5. Fragmenty
6. UI
7. REST oraz OAuth2
8. Serwisy
9. Notyfikacje
10. Wydajność

11. OpenGL, oraz VR
12. Wearables, NDK, IoT
13. Google Play
14. React Native



Zaliczenie

Średnia ocena ze wszystkich projektów.

Warunki:

- Termin na każdy projekt: max. 2 tygodnie po zajęciach
- Każdy projekt zaliczony na minimum 3.0

Egzamin: test wyboru, 30 pytań. Aby zaliczyć trzeba poprawnie odpowiedzieć na 20 pytań.

Terminy: 1.02 oraz 15.02.

Kotlin



Trochę historii

Powstał jako (szybsza) alternatywą dla Scali w 2011 roku.

Twórcą języka jest Dmitry Jemerov, a firma która rozwija język to JetBrains.

Od 2017 jest wspieranym językiem na platformie Android.

Obecnie jedynie nieliczni decydują się na Javę pisząc aplikację na Androida.



Charakterystyka języka

Działa na JVM

Podobnie jak Scala, pisanie kodu w Kotlinie jest krótsze i szybsze

Statycznie typowany

**Zanim przejdziemy do pisania
aplikacji na platformę Android**



Garbage collector

Języki, które go stosują to m.in.: Java oraz inne na JVM, JavaScript, C#, Go, Ruby.

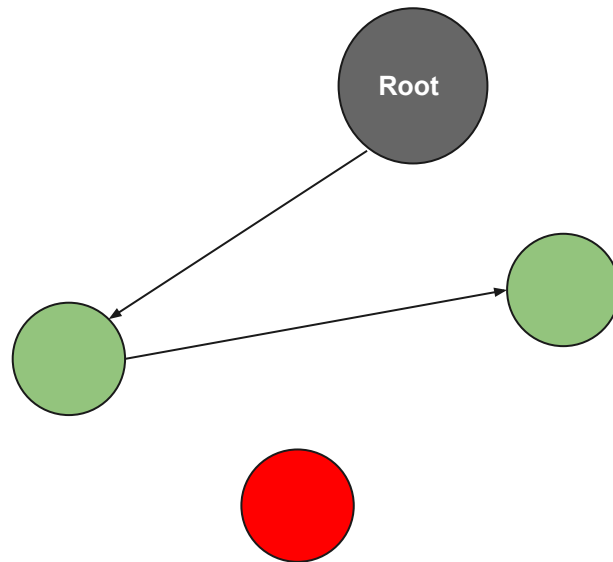
Istnieje wiele implementacji GC:

- Tracing GC - składa się z dwóch etapów: wykrywania „żyjących” obiektów oraz zwalnianie pamięci,
- RC GC - podobne do ARC; posiada osobny proces GC, który czyści pamięć

Garbage collector

Mark - zaznacz (zielone)

Sweep - usuń (czerwone)





Java Garbage collector

Istnieje wiele GC w Javie 11 (-XX:):

- Serial Collector - jeden wątek, dobry dla małych aplikacji
- Parallel Collector - wiele wątków; średnie aplikacje
- Garbage-First Collector - równoległe wątki; wykorzystywany na sprzęcie z wieloma procesorami i dużą ilością pamięci
- Z Garbage Collector - dostępny od Javy 11 zoptymalizowany na ograniczenie opóźnień po stronie aplikacji.

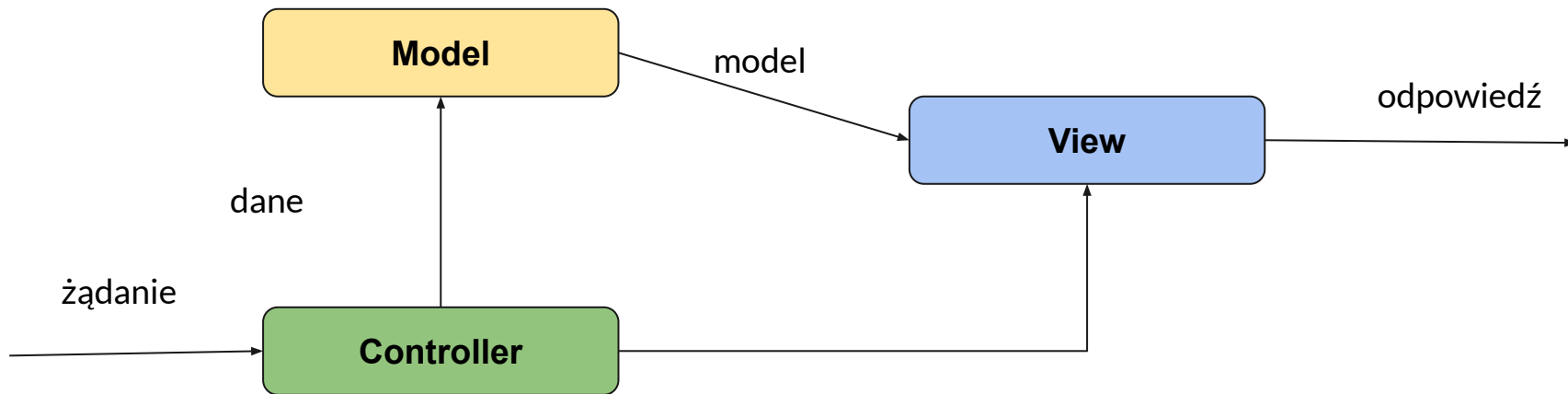



Zarządzanie pamięcią w różnych językach

- Python - wykorzystuje połączenie RC z GC.
- Java - GC
- JavaScript - GC
- Haskell - GC
- Go - GC via Go scheduler
- Ruby - GC
- Swift - ARC
- Lisp - GC
- COBOL - manualne
- Lua - GC



MVC





MVC - zalety

Logika biznesowa oddzielona jest od widoku,

Nie ma zależności modelu od widoku

Uporządkowany kod



MVC - wady

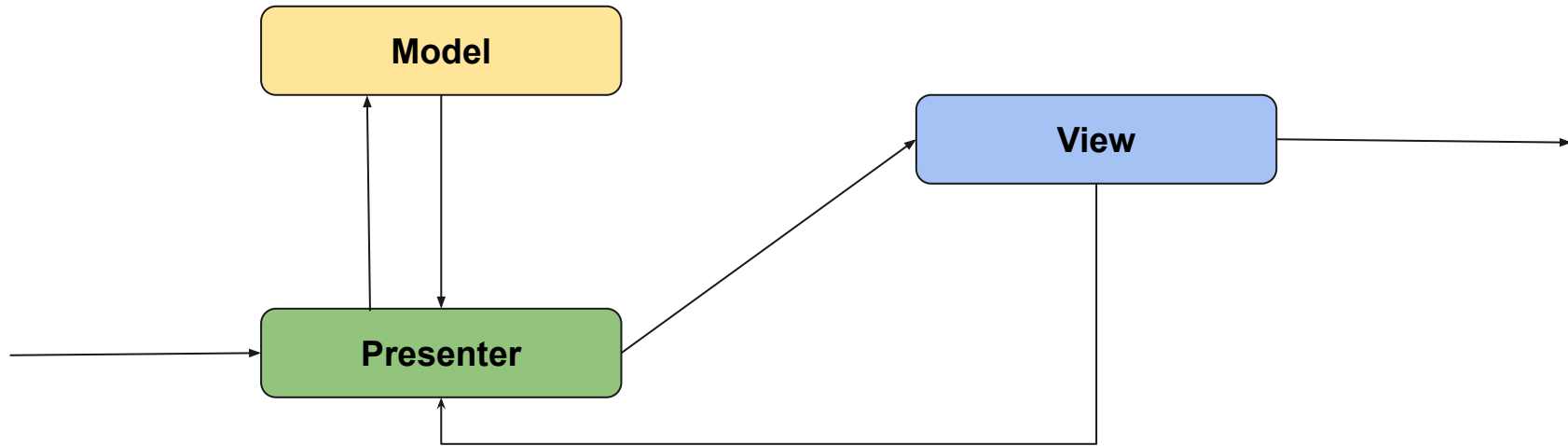
Złożoność aplikacji


Widok zależny od modelu

Widoki są trudne w testowaniu



MVP





MVP - zalety

Pochodna MVC

Oddzielona logika od widoku

Brak zależności modelu od widoku,

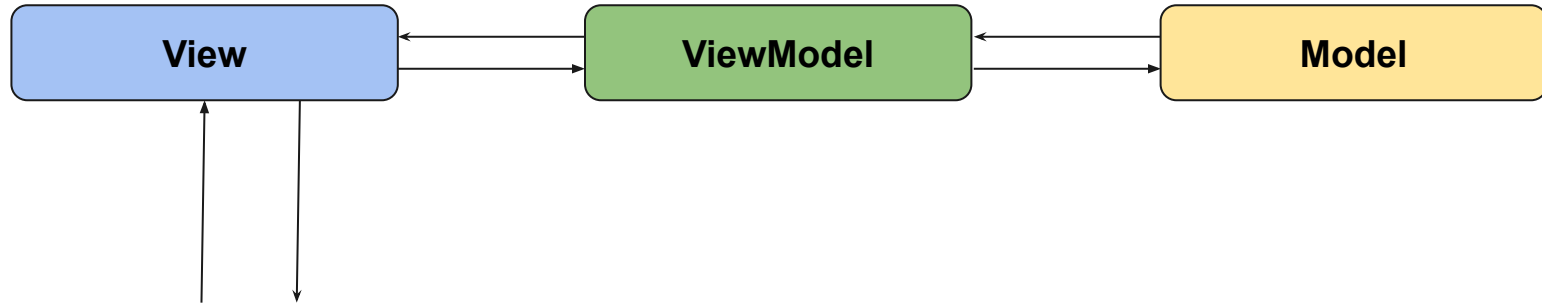



MVP - wady

Skomplikowana dla prostych aplikacji



MVVM






MVVM- zalety

Podział na widok i ViewModel

Testowanie jest prostsze niż w MVC

Asynchroniczność



MVVM - wady

ViewModel nie komunikuje się z warstwą widoku

Dużo klas, ponieważ każdy widok ma więcej niż jedną klasę po stronie widoku i ViewModel.



Wzorce projektowe

Jest kilka grup wzorców projektowych:

- strukturalne
 - Adapter
 - Most (Bridge)
 - Kompozyt (Composite)
 - Dekorator (Decorator)
 - Fasada (Facade)
 - Pyłek (Flyweight)
 - Pełnomocnik (Proxy)



Wzorce projektowe

Jest kilka grup wzorców projektowych:

- strukturalne
- kreacyjne
- Fabryka abstrakcyjna (Abstract Factory)
- Budowniczy (Builder)
- Metoda wytwórcza (Factory Method)
- Object Pool (Pula obiektów)
- Prototyp (Prototype)
- Singleton



Wzorce projektowe

Jest kilka grup wzorców projektowych:

- strukturalne
 - kreacyjne
 - behawioralne
- Łańcuch zobowiązań (Chain of responsibility)
 - Polecenie (Command)
 - Interpreter
 - Iterator
 - Mediator
 - Memento
 - Obserwator (Observer)
 - Stan (State)
 - Strategia (Strategy)
 - Metoda szablonowa (Template Method)
 - Wizytator (Visitor)



Kreacyjne wzorce projektowe



Fabryka abstrakcyjna (Abstract factory)

Założenia

- Dostarcza interfejs do tworzenia całej rodziny powiązanych obiektów bez specyfikowania klasy z implementacją
- Hierarchia, która enkapsuluje wiele klas (produktów)

Przykład

Maszyny produkujące elementy samochodów wykorzystują ten wzorzec. Mogą po podmianie niektórych części maszyny, mogą one produkować różne elementy samochodu do różnych ich modeli.



Budowniczy (Builder)

Założenia

- Oddziela logikę odpowiedzialną za tworzenie złożonych obiektów od ich reprezentacji
- Umożliwia na tworzenie różnych obiektów w zależności ich reprezentacji

Przykład

Tworzenie różnych rodzajów pizzy jest takim przykładem. Główny proces jest taki sam, różnią się jedynie



Fabryka (Factory method)

Założenia

- Definiuje interfejs dla tworzenia obiektu, jednocześnie pozostawia decyzję klasie, którą klasę zainicjalizować
- W odróżnieniu od budowniczego skupia się wokół konstruktora, jest prostsza.

Przykład

Jest wykorzystywana przez budowniczego.



Pula obiektów (Object pool)

Założenia

- Ma również zastosowanie przy projektach, gdzie istotna jest prędkość działania
- Pozwala na alokowanie pamięci dla grupy obiektów, ponieważ ta część jest najbardziej czasochłonna
- Udostępnia instancje obiektów

Przykład

W grach jest często wykorzystywana oraz wszędzie tam, gdzie wykorzystywane są identyczne obiekty wiele razy.



Prototyp (Prototype)

Założenia

- Jest prototypem obiektu
- W wielu językach klonuje albo kopiuje się prototyp

Przykład

W języku JavaScript jest podstawowym pojęciem.



Singleton

Założenia

- Jeden z prostszych wzorców
- Zapewnia istnienie jednej instancji danej klasy

Przykład

Wykorzystywany jest często w autoryzacji czy dostępie do bazy danych.



Strukturalne wzorce projektowe



Adapter

Założenia

- Konwertuje interfejs klasy na interfejs, który jest oczekiwany przez klienta
- Pozwala na pracę pomiędzy klasami, które normalnie nie mogłyby pracować
- Pozwala na współpracę starego/innego rozwiązania z naszym

Przykład

Brak kompatybilności pomiędzy standardami obsługi odpowiedzi od serwera. Do obsługi możemy wykorzystać interfejs, który pozwoli na obsługę konkretnej odpowiedzi z serwera przez klasę, która nie została zaimplementowana do obsługi odpowiedzi danego typu. Innym przykładem jest wtyczka do prądu w UK, US oraz w Europie. Istnieją adaptery, które pozwalają na pracę wtyczek z kontaktami w różnych krajach.



Most (Bridge)

Założenia

- Celem jest rozdzielenie abstrakcji od implementacji
- Udostępnia hierarchię interfejsów oraz odpowiednią hierarchię implementacji

Przykład

Most rozbija część implementacji od jej abstrakcji, w taki sposób że są od siebie niezależne. Przykładem może być przełącznik do włączania/wyłączania światła. Abstrakcja jest jedna, a implementacji wiele.



Kompozyt (Composite)

Założenia

- Kompozyt tworzy reprezentację obiektów za pomocą drzewa. Jednocześnie kompozyt pozwala na dostęp do poszczególnych elementów ze względu na dziedziczenie po interfejsie nadrzędnym
- Upraszcza kod aplikacji
- Rekursywne

Przykład

Operacja arytmetyczna może być kompozytem, ponieważ $(2+3) - (6+2)$ jest tym samym co $2+3-6+2$.



Dekorator (Decorator)

Założenia

- Dodaje dodatkowe funkcjonalności do obecnie istniejących klas opakowując je, wywołując metody wewnątrz
- Jest alternatywą dla tworzenia podklas

Przykład

Dodanie kilku klas CSS do zwracanego przez klasę kodu HTML jest dobrym przykładem dekoratora.



Fasada (Facade)

Założenia

- Opakowuje złożoną część kodu przez proste interfejsy
- Uogólnia system za pomocą prostych interfejsów
- Wysokopoziomowe podejście

Przykład

Fasadą jest niemal każdy call center. Dzwonimy na numer biura obsługi klienta i wybieramy kolejno numery, aby skontaktować się z odpowiednią osobą, która jest odpowiednia dla rozwiązania naszego problemu.



Pyłek (Flyweight)

Założenia

- Pozwala na współdzielenie bardzo wielu małych obiektów
- Ma zastosowanie w usprawnieniach wydajności

Przykład

Przeglądarki wykorzystują ten wzorzec do ładowania dużej liczby rysunków.



Proxy

Założenia

- Tworzy obiekt pośredni w komunikacji pomiędzy dwoma innymi
- Pozwala na dostęp oraz kontrolę drugim obiektem

Przykład

Najprostszy przykładem jest proxy wykorzystywane w przeglądarkach.



Funkcyjne wzorce projektowe



Monoid

Założenia

- Struktura algebraiczna
- Występuje funkcja zwracająca wartość pustą oraz metoda łącząca zwracająca wartość podaną jako argument, np. combine

Przykład

MapReduce wykorzystuje monoidy. Option jest monoidem



Monada

Założenia

- Pozwala na zastąpienie wielu wywołań, które od siebie zależą funkcjami unit oraz bind
- Upraszcza operacje pomiędzy funkcjami zależnymi
- Jest monoidem

Przykład

Monada zamienia zagnieżdżone wyrażenia w proste (flatten)



Funktor

Założenia

- Funktor stosuje operację na elementach listy, funkcjach lub innych strukturach
- Zawiera funkcję mapującą (map, flatmap),
- Wyjście jest typu wejścia

Przykład

Funktory są często wykorzystywane przy listach, np. kiedy je mapujemy.



Behawioralne wzorce projektowe



Łańcuch odpowiedzialności (Chain of responsibilities)

Założenia

- Przekazuje obiekt do kolejnych obiektów w łańcuchu, aż któryś z nich je obsłuży

Przykład

Przykładem jest każdy framework MVC albo bankomat.



Komenda (Command)

Założenia

- Enkapsuluje wywołanie/żądanie w obiekcie, jednocześnie parametryzując żądanie w obiekcie

Przykład

Przykładem jest PyCall, który wywołuje komendę w Pythonie z poziomu języka Ruby.



Interpreter

Założenia

- Mapuje gramatykę języka i interpretuje ją w celu wykonania poszczególnych komend

Przykład

Występuje w każdym języku skryptowym



Iterator

Założenia

- Pozwala na łatwe przejście sekwencyjne przez kolekcję

Przykład

Występuje w większości kolekcji w językach obiektowych.



Mediator

Założenia

- Pozwala na komunikację (kontrolę) się wielu niezwiązanych ze sobą bezpośrednio obiektów.
- Pozwala na komunikację wielu-do-wielu

Przykład

Przykładem jest wieża kontroli lotów.



Memento

Założenia

- Pozwala na zapamiętanie stanu obiektu, dzięki czemu może wrócić do poprzedniego stanu obiektu w przypadku błędów
- Może tworzyć historię zmian i odtwarzać stany z przeszłości (check points)

Przykład

Niektóre debugery realizują ten wzorec i pozwalają na powrót stanu poprzedniego.



Obserwator (Observer)

Założenia

- Potrafi zdefiniować zależność jeden do wielu, w taki sposób, że gdy jeden z obiektów zmieni swój stan, pozostałe obiekty są o tym informowane

Przykład

View w MVC oraz aukcje.



Stan (State)

Założenia

- Podobnie jak obserwator, ale dotyczące jednego obiektu
- Obiekt zmieniając swój stan zmienia swoje zachowanie
- Obiektowe maszyna stanów

Przykład

Maszyna vendingowa, chatboty



Strategia (Strategy)

Założenia

- Definiuje wiele metod do rozwiązania podobnego problemu w różny sposób.
- Strategia je enkapsuluje i pozwala na wybór odpowiedniej metody w zależności od wejścia

Przykład

Wybór trasy przejazdu z punktu A do B.



Metoda szablonowa (Template method)

Założenia

- Tworzy szablon/szkielet metody, w której niektóre kroki są wywołaniem metod z innych subklas.
- Klasa bazowa określa gdzie znajdują się implementacje poszczególnych kroków metody

Przykład

Budowanie domu.



Wizytator (Visitor)

Założenia

- Wizytator pozwala na zdefiniowanie operacji bez zmiany danej klasy oraz elementów danej klasy na której operuje.
- Pozwala na odzyskanie utraconych informacji

Przykład

Taksówka.



Wzorce projektowe - porównanie



Budowniczy vs Fabryka

Budowniczy

- Zawiera metodę `build()`, która buduje instancję, ale parametry mogą być podawane przez settery
- Ma najczęściej wiele metod, które rozkładają tworzenie obiektu na wiele etapów (funkcjonalności)

Fabryka

- Prostsza od budowniczego
- Wymaga podania wszystkich parametrów w jednej metodzie



Most vs Strategia

Most

- Tworzymy strukturę interfejsów
- Wykorzystywane, gdy chcemy użyć tych interfejsów w innym celu
- Wybór „opcji” wybierany jest na poziomie implementacji

Strategia

- Zmiana implementacji podczas wykonywania kodu, tj. wybieramy, który obiekt wykona daną akcję



Fasada vs Dekorator

Fasada

- Nie dodaje dodatkowych funkcjonalności do istniejącego obiektu

Dekorator

- Dodaje dodatkowych funkcjonalności do istniejącego obiektu



Środowisko oraz kilka przykładów

Wprowadzenie do platformy Android





SDK

SDK Tools - narzędzia do tworzenia aplikacji na Andorida

SDK Platform - przede wszystkim kod źródłowy (wersja API)

SDK Build tools - narzędzia do budowania aplikacji na Androida

SDK Platform tools - narzędzia, które wykorzystują SDK Platform, np. adb



Andorid NDK

Native Development Kit - pozwala na budowanie aplikacji na Androida w C/C++.

<https://developer.android.com/ndk>



Android inaczej

JavaScript

<https://reactnative.dev/>

Dart

<https://flutter.dev/>

Python

<https://beeware.org/>



Budowanie z linii poleceń

Budowanie:

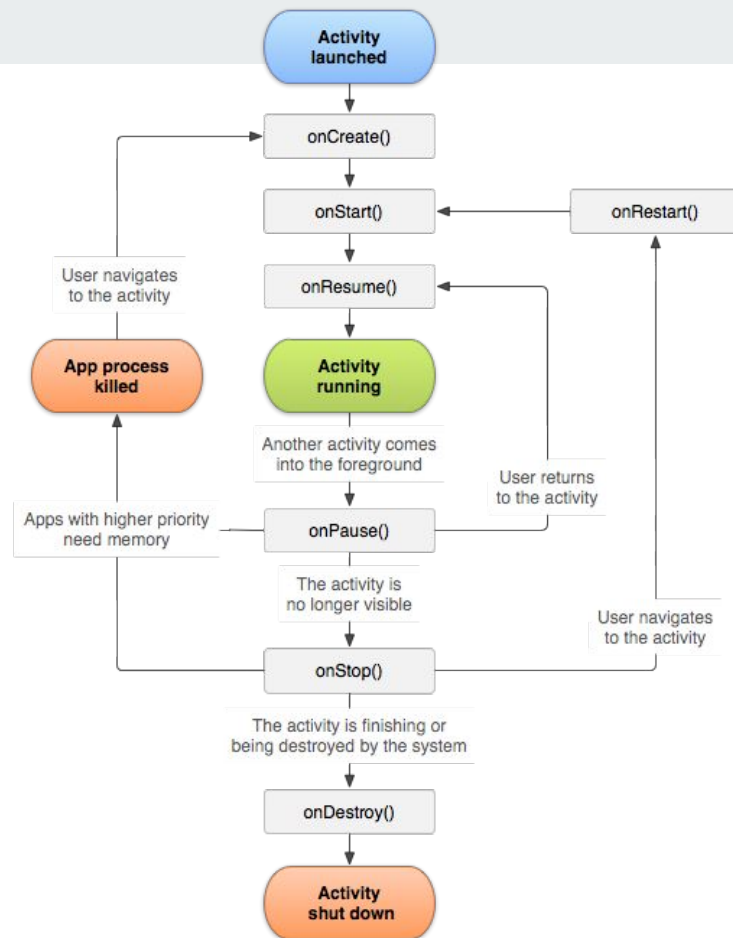
```
./gradlew assembleDebug
```

Instalowanie:

```
./gradlew installDebug
```

```
adb install .apk
```

Aktywności





Intencje (intent)

Łączy elementy w aplikacji Androidowej ze sobą.

Zawiera:

- akcję
- dane

Przykład: dwa widoki



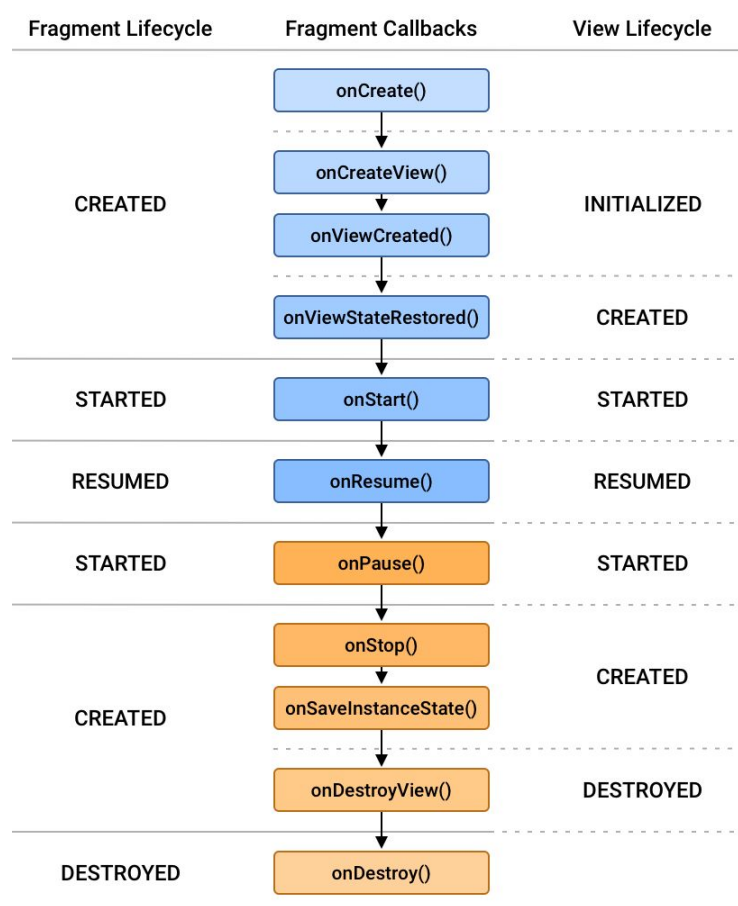
Komponenty

Grupy komponentów:

- tekstowe
- przyciski
- widżety
- layouty
- kontenery
- helpery
- googlowskie

Fragmenty

Cykl życia fragmentu



Źródło: <https://developer.android.com/guide/fragments/lifecycle>



Fragmenty

FragmentManager

Fragment

FragmentManager

Networking





Networking

```
<uses-permission android:name="android.permission.INTERNET" />
```



Networking configuration

```
<application android:networkSecurityConfig="@xml/network_security_config" />
```

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config>
    <domain includeSubdomains="true">secure.example.com </domain>
    <domain includeSubdomains="true">cdn.example.com </domain>
    <trust-anchors>
      <certificates src="@raw/trusted_roots" />
    </trust-anchors>
    <domain-config cleartextTrafficPermitted="false">
      <domain>developer.android.com </domain>
    </domain-config>
  </domain-config>
</network-security-config>
```

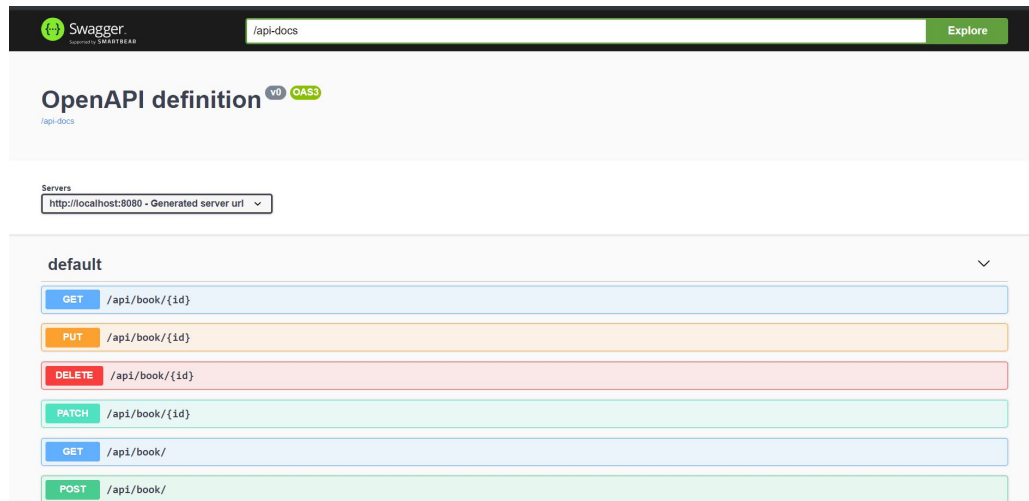
Swagger

<https://openapi-generator.tech/docs/generators/kotlin/>

<https://ktor.io/docs/openapi-swagger.html>

<https://github.com/nielsfalk/ktor-swagger>

<https://github.com/papsign/Ktor-OpenAPI-Generator>



Źródło: <https://www.baeldung.com/spring-rest-openapi-documentation>



Couroutines a Wątki

Wątki - OS

Wątki - duże

Wątki - zawsze te same

Couroutine - Kotlin Runtime

Couroutine - zabierają mniej zasobów

Couroutine - mogą być najpierw jednym, a później kolejnym wątku obsługiwane



Coroutines

```
fun main() = runBlocking {  
    launch {  
        delay(1000L)  
        println("World!")  
    }  
    println("Hello")  
}
```



Coroutines

`-Dkotlinx.coroutines.debug`



Coroutines - Debugowanie

`-Dkotlinx.coroutines.debug`



Couroutines

```
fun main() =  
    runBlocking {  
        launch { doWorld()  
    }  
    println("Hello")  
}
```

```
suspend fun doWorld() {  
    delay(1000L)  
    println("World!")  
}
```




Couroutines

```
fun main() =  
    runBlocking {  
        launch { doWorld()  
    }  
    println("Hello")  
}
```

```
suspend fun doWorld() {  
    delay(1000L)  
    println("World!")  
}
```



Coroutines

```
suspend fun doWorld() = coroutineScope {  
    launch {  
        delay(1000L)  
        println("World!")  
    }  
    println("Hello")  
}
```



Coroutines

```
suspend fun doWorld() = coroutineScope {  
    launch {  
        delay(2000L)  
        println("World 2")  
    }  
}
```

```
launch {  
    delay(1000L)  
    println("World 1")  
}  
println("Hello")  
}
```



Couroutines

```
val job = launch {  
    delay(1000L)  
    println("World!")  
}  
  
println("Hello")  
  
job.join()  
  
println("Done")
```



Couroutines w Retrofit

```
val service = RetrofitService.create()

val call = service.postProductCall(product)

call.enqueue(object : Callback<Unit> {

    override fun onResponse(call: Call<Unit>, response: Response<Unit>) {

        Log.d("POST_PRODUCT", "Product post successful")

    }

    override fun onFailure(call: Call<Unit>, t: Throwable) {

        Log.d("POST_PRODUCT", t.message.toString())

    }

})
```



Couroutines

```
val job = launch {  
    repeat(1000) { i ->  
        println("job: I'm sleeping $i ...")  
        delay(500L)    }}  
  
delay(1300L)  
  
println("main: I'm tired of waiting!")  
  
job.cancel()  
  
job.join() // job.cancelAndJoin()  
  
println("main: Now I can quit.")
```



Coroutines

```
suspend fun doSomethingUsefulOne(): Int {  
    delay(1000L)  
    return 13  
}
```

```
suspend fun doSomethingUsefulTwo(): Int {  
    delay(1000L)  
    return 29  
}
```

```
val time = measureTimeMillis {  
    val one = doSomethingUsefulOne()  
    val two = doSomethingUsefulTwo()  
    println("The answer is ${one + two}")  
}  
  
println("Completed in $time ms")
```



Couroutines Dispatcher

Unconfined : I'm working in thread main

Default : I'm working in thread DefaultDispatcher-worker-1

newSingleThreadContext: I'm working in thread MyOwnThread

main runBlocking : I'm working in thread main



Retrofit

1. It is very fast.
2. It enables direct communication with the web service.
3. It is easy to use and understand.
4. It supports request cancellation.
5. It supports post requests and multipart uploads.
6. It supports both synchronous and asynchronous network requests.
7. Supports dynamic URLs.
8. Supports convertors.



Retrofit

```
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.0'
```

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
```

```
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
```



Retrofit

```
interface RetrofitService {  
  
    @GET("product")  
  
    fun getProductsCall() : Call<List<Product>>  
  
  
    @POST("customer")  
  
    fun postCustomerCall(@Body customer : Customer) : Call<Unit>
```



Retrofit

```
@FormUrlEncoded
```

```
@POST("user/edit")
```

```
Call<User> updateUser(@Field("first_name") String first, @Field("last_name") String last);
```

```
@Multipart
```

```
@PUT("user/photo")
```

```
Call<User> updateUser(@Part("photo") RequestBody photo, @Part("description") RequestBody description);
```



Retrofit

```
companion object {  
    //    var BASE_URL = "https://74ce-185-25-121-195.ngrok.io/"  
    var BASE_URL = "http://10.0.2.2:8080/"  
  
    fun create() : RetrofitService {  
        val retrofit = Retrofit.Builder()  
            .addConverterFactory(GsonConverterFactory.create())  
            .baseUrl(BASE_URL)  
            .build()  
        return retrofit.create(RetrofitService::class.java)  
    }  
}
```



Retrofit

```
val service = RetrofitService.create()

val call = service.deleteAllCustomersCall()

call.enqueue(object : Callback<Unit> {

    override fun onResponse(call: Call<Unit>, response:
Response<Unit>) {

        if(response.isSuccessful)

            Log.d("DELETE_ALL_CUSTOMERS" , "success")

    }

    override fun onFailure(call: Call<Unit>, t:
Throwable) {

        Log.d("DELETE_ALL_CUSTOMERS" ,
t.message.toString())

    }

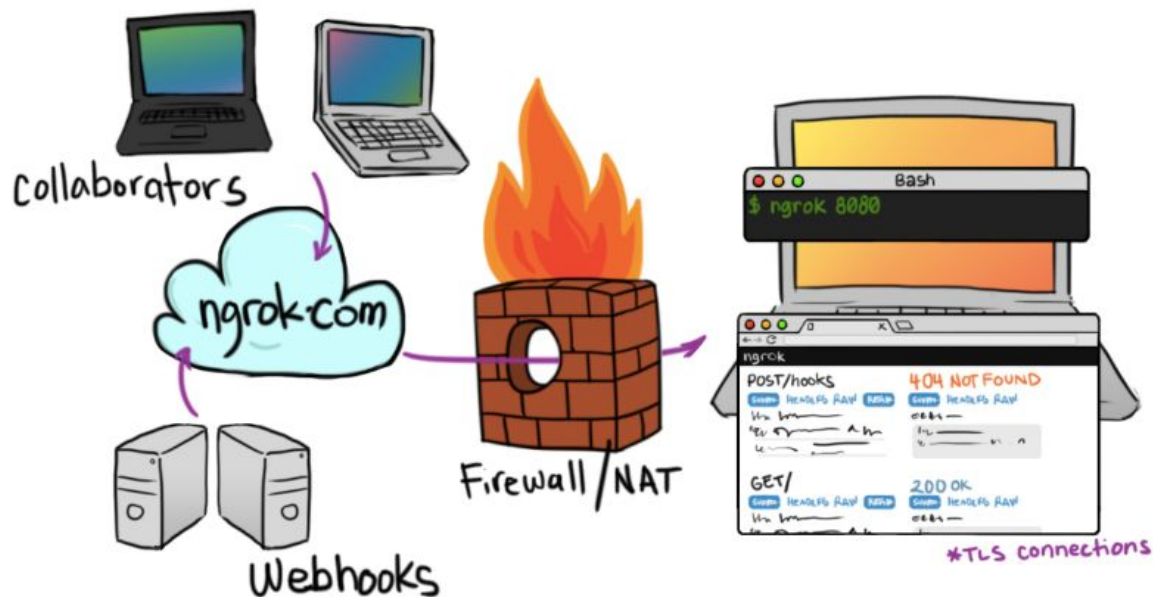
})
```



Retrofit - alternatywy

Volley - <https://google.github.io/volley/>

Ngrok





Ngrok

```
$ ngrok http 8080
```

Session Status online

Session Expires 1 hour, 59 minutes

Terms of Service <https://ngrok.com/tos>

Version 3.1.0

Region Europe (eu)

Latency -

Web Interface <http://127.0.0.1:4041>

Forwarding <https://9373-2a01-115f-440b-7200-34c6-284-a4f9-c1f>

Connections	ttl	opn	rt1	rt5	p50	p90
0	0	0.00	0.00	0.00	0.00	

OAuth2

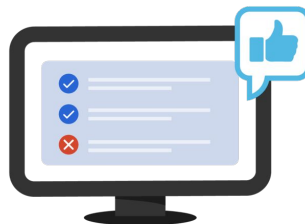
OAuth2

Authentication



Confirms users
are who they say they are.

Authorization



Gives users permission
to access a resource.

okta

Źródło:

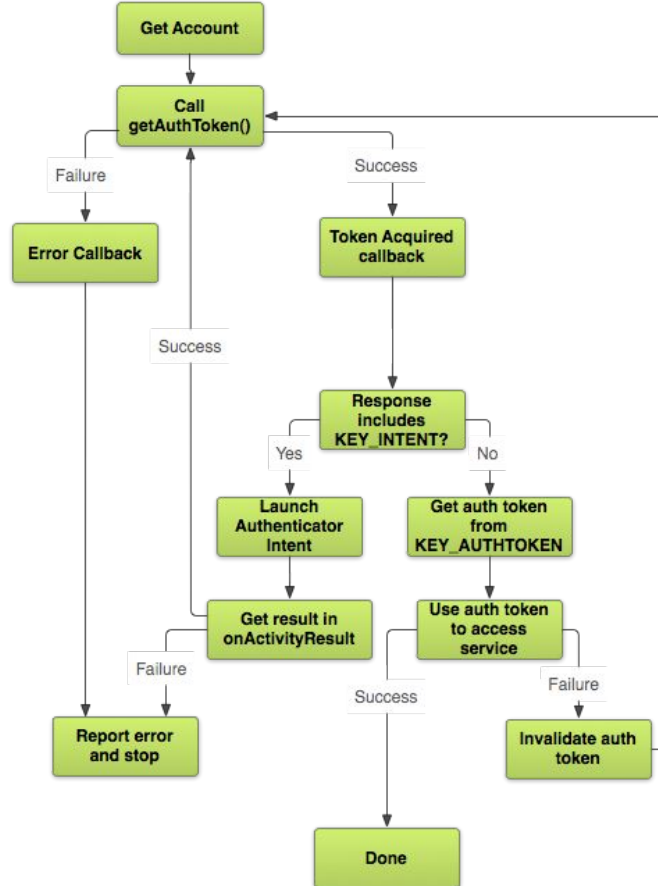
[https://www.okta.com/identity-101/authentication-vs-authorization/#:~:text=Authentication%20confirms%20that%20users%20are,and%20access%20management%20\(IAM\).](https://www.okta.com/identity-101/authentication-vs-authorization/#:~:text=Authentication%20confirms%20that%20users%20are,and%20access%20management%20(IAM).)



OAuth2 - alternatywy

OpenID Connect, Auth0, JSON Web Token, Keycloak, Amazon Cognito

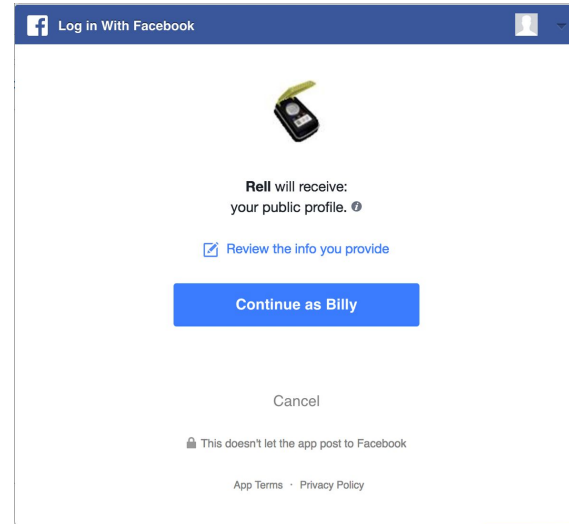
OAuth2





OAuth2


<https://developers.facebook.com/docs/facebook-login/guides/advanced/manual-flow>





OAuth2

<https://docs.github.com/en/developers/apps/building-oauth-apps/authorizing-oauth-apps>



Device Activation

Enter the code displayed on your device

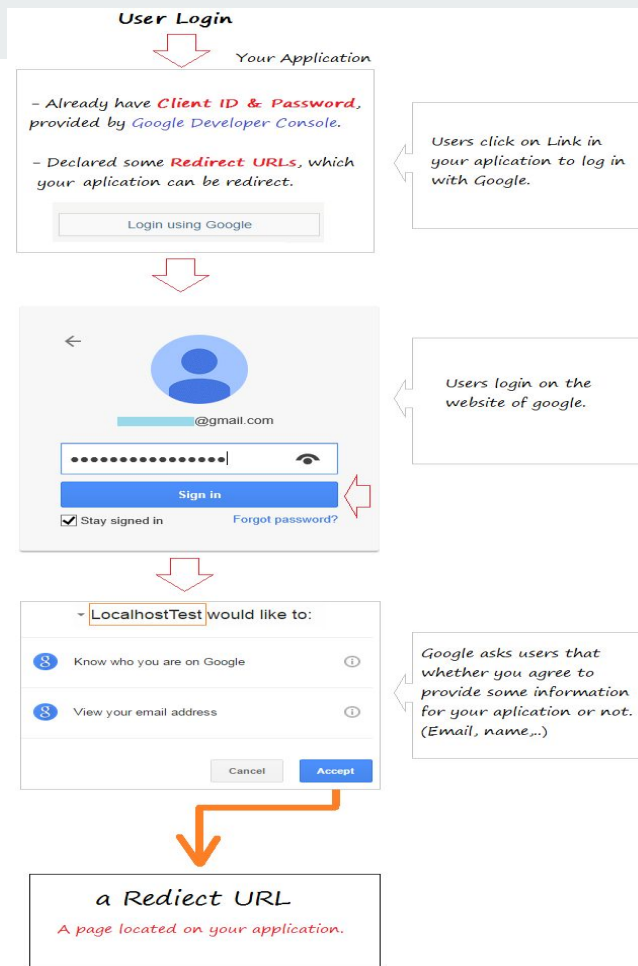
-

Continue

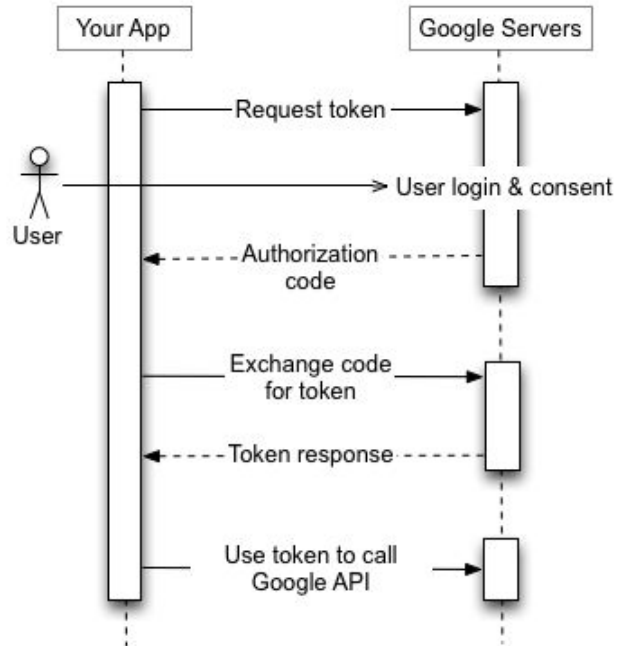
GitHub staff will never ask you to enter your code on this page.

OAuth2

<https://docs.github.com/en/developers/apps/building-oauth-apps/authorizing-oauth-apps>



OAuth2





OAuth2

```
val am: AccountManager = AccountManager.get(this)
```



OAuth2

```
val url =  
URL("https://www.googleapis.com/tasks/v1/users/@me/lists?key=$  
your_api_key")  
val conn = url.openConnection() as HttpURLConnection  
conn.apply {  
    addRequestProperty("client_id", your client id)  
    addRequestProperty("client_secret", your client secret)  
    setRequestProperty("Authorization", "OAuth $token")  
}
```



OAuth2

```
val url =  
URL("https://www.googleapis.com/tasks/v1/users/@me/lists?key=$  
your_api_key")  
val conn = url.openConnection() as HttpURLConnection  
conn.apply {  
    addRequestProperty("client_id", your client id)  
    addRequestProperty("client_secret", your client secret)  
    setRequestProperty("Authorization", "OAuth $token")  
}
```



OAuth2

<https://github.com/openid/AppAuth-Android>

Struktura



ToDo

<https://github.com/android/architecture-samples/tree/todo-mvp-kotlin/todoapp>



Firestore

<https://github.com/firebase/quickstart-android/tree/master/database>



Realm

<https://github.com/realm/realm-kotlin-samples/tree/main/Bookshelf/androidApp>



MVVM

<https://github.com/ianishar/android-mvvm-architecture>

<https://github.com/PatilShreyas/Foodium>

<https://github.com/androiddevnotes/awesome-android-kotlin-apps>

Płatności



Stripe

<https://github.com/stripe/stripe-android>



Stripe - backend

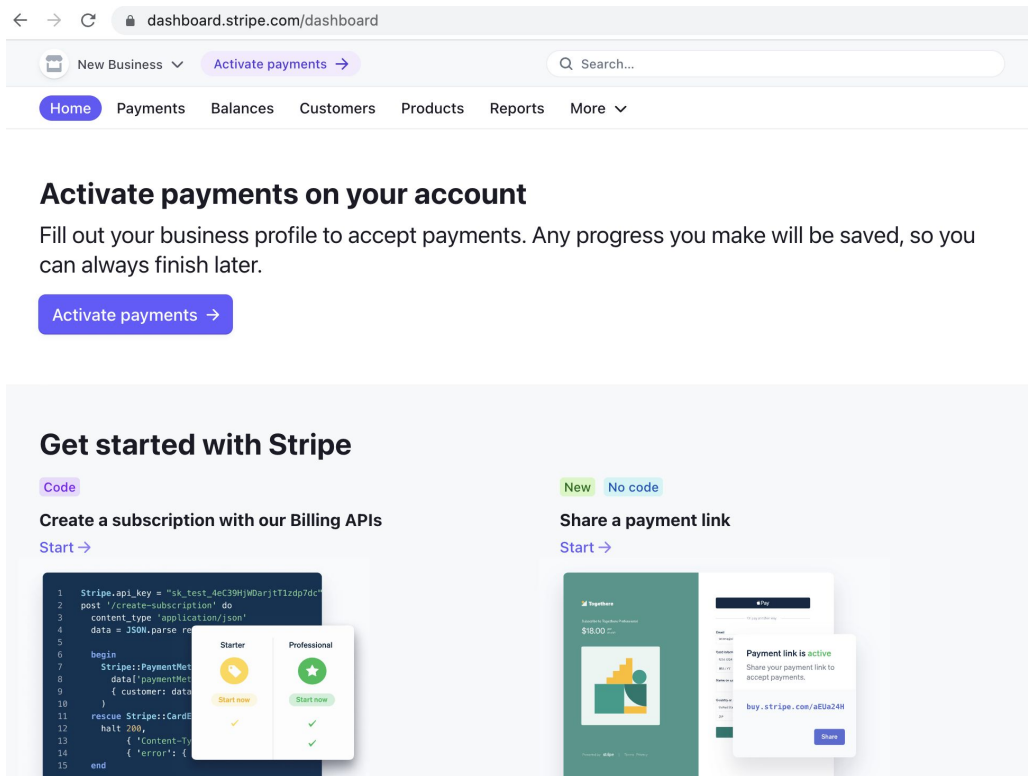
```
implementation 'com.stripe:stripe-java:22.0.0'
```



Stripe - Android

```
implementation 'com.stripe:stripe-android:20.16.1'
```

Stripe - Dashboard



The screenshot displays the Stripe Dashboard interface. At the top, the browser address bar shows 'dashboard.stripe.com/dashboard'. Below the address bar, there's a navigation bar with a 'New Business' dropdown, an 'Activate payments' button with a right arrow, and a search bar. A secondary navigation bar contains links for 'Home', 'Payments', 'Balances', 'Customers', 'Products', 'Reports', and 'More' with a dropdown arrow. The main content area is titled 'Activate payments on your account' and includes a sub-header 'Fill out your business profile to accept payments. Any progress you make will be saved, so you can always finish later.' Below this is a prominent 'Activate payments' button. Further down, the 'Get started with Stripe' section is visible, featuring two options: 'Code' (with a 'Create a subscription with our Billing APIs' link and a 'Start' button) and 'No code' (with a 'Share a payment link' link and a 'Start' button). The 'Code' option shows a code snippet for creating a subscription and a comparison of 'Starter' and 'Professional' plans. The 'No code' option shows a payment link interface with a 'Payment link is active' confirmation and a 'Share' button.

← → ↻ dashboard.stripe.com/dashboard

New Business ▾ Activate payments → 🔍 Search...

Home Payments Balances Customers Products Reports More ▾

Activate payments on your account

Fill out your business profile to accept payments. Any progress you make will be saved, so you can always finish later.

Activate payments →

Get started with Stripe

Code

Create a subscription with our Billing APIs

Start →

```
1 Stripe.api_key = "sk_test_4eC39HjWoarj1T1dp7dC"
2 post "/create-subscription" do
3   content_type "application/json"
4   data = JSON.parse request.body
5
6   begin
7     Stripe::PaymentMethod.create(
8       data: { payment_method: { type: "card", card: { number: "4242 4242 4242 4242", exp_year: [2016, 2017], cvc: "123" } } }
9     )
10
11   rescue Stripe::CardError
12     halt 200, { "Content-Type" => "text/plain", "error" => { "message" => "Invalid card number" } }
13   end
14 end
```

Starter Professional

Start now Start now

New No code

Share a payment link

Start →

Payment link is active

Share your payment link to accept payments.

buy.stripe.com/aEla24H

Share



PayU

```
maven { url "https://payu.jfrog.io/payu/mobile-sdk-gradle-local" }
```

```
mavenLocal()
```

```
implementation "com.payu.android.front.sdk:{MODULE_NAME}:{LIBRARY_VERSION}"
```




Library	MODULE_NAME
Choosing Payment Method	payment-library-chooser-module
Card Scanner	payment-library-card-scanner
Google Pay	payment-library-google-pay-module
Google Pay adapter	payment-library-google-pay-adapter
Add Card	payment-add-card-module
WebView	payment-library-webview-module

Jakość



Proces testowania (ang. test process)

Proces testowania składa się z kilku części:

- planowania i zarządzania testami,
- analizy i projektowanie testów,
- implementacja oraz wykonywanie testów,
- ewaluacji kryteriów wyjściowych (ang. exit criterias) oraz raportowania wyników,
- wykonywanie procesów końcowych testów (ang. test closure).



Przypadki testowe (ang. test case)

Przypadek testowy składa się z listy kroków z akcjami oraz oczekiwanym wynikiem każdej akcji. Na przykład, gdy chcemy przetestować autoryzację można stworzyć przypadek jak poniżej.

Krok	Akcja	Oczekiwany wynik
1	Przejdź na stronę: <code>http://localhost/login</code>	Formularz logowania jest widoczny z polami: <i>username</i> oraz <i>password</i> .
2	Wypełnij pola <i>username</i> oraz <i>password</i> danymi: <i>admin</i> oraz <i>secret</i> .	Pole <i>username</i> zostało wypełnione jawnym tekstem. Pole <i>password</i> zostało wypełnione tekstem niejawnym.
3	Kliknij przycisk <i>login</i>	Formularz został wysłany
4	Sprawdź obecną stronę po przekierowaniu	Strona po przekierowaniu to: <code>http://localhost/main</code>

Wykonywanie przypadków testowych jest proste, ponieważ oczekiwane wyniki są jasno zdefiniowane.

Przypadki testowe mogą zostać wykorzystane do testów regresyjnych czy akceptacyjnych.



Przypadki testowe

Często jednak przypadki testowe pozwalają również pokryć negatywne scenariusze, gdzie wynikiem jest błąd.

Krok	Akcja	Oczekiwany wynik
1	Przejdź na stronę: <code>http://localhost/login</code>	Formularz logowania jest widoczny z polami: <i>username</i> oraz <i>password</i> .
2	Wypełnij pola <i>username</i> oraz <i>password</i> danymi: <i>admin</i> oraz <i>secret</i> .	Pole <i>username</i> zostało wypełnione jawnym tekstem. Pole <i>password</i> zostało wypełnione tekstem niejawnym.
3	Kliknij przycisk <i>login</i>	Formularz został wysłany
4	Sprawdź obecną stronę po przekierowaniu	Strona po przekierowaniu to: <code>http://localhost/main</code>
5	Sprawdź stronę z błędem (popup)	Wyświetlony został komunikat z błędem: "Błędny użytkownik/hasło. Spróbuj ponownie."



Zestawy testów oraz pokrycie testami

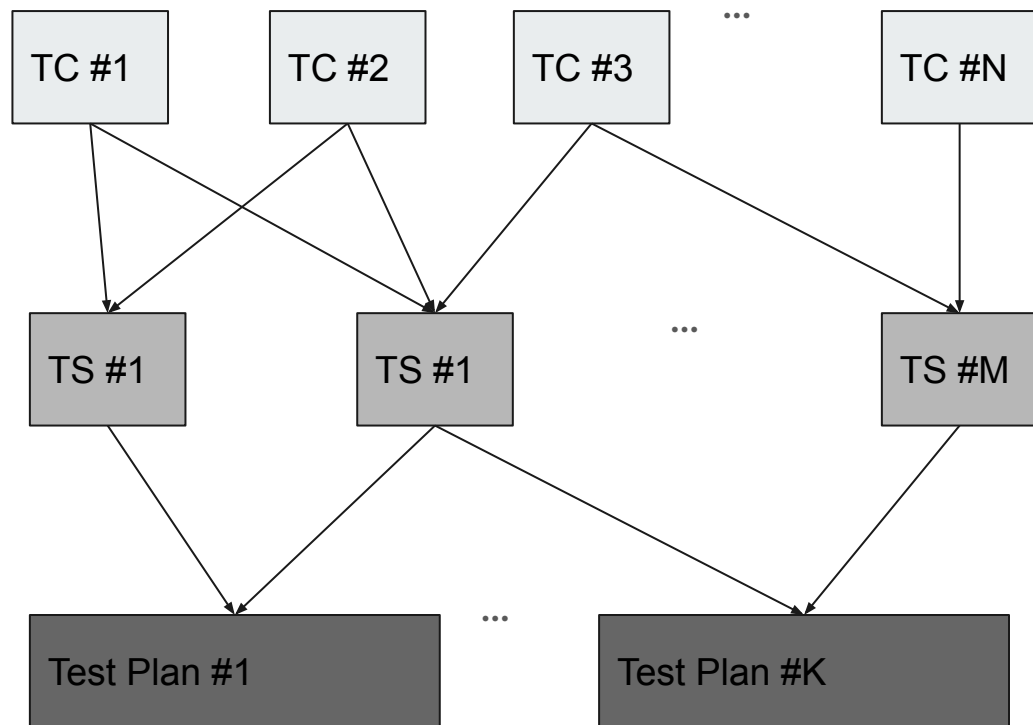
Zestaw testów (ang. test suites) składają się z przypadków testowych. Zestawy zawierają przypadki, które testują konkretną funkcjonalność.

Pokrycie testami (ang. test coverage) jest określone procentowo w jakim stopniu testy pokrywają kod źródłowy. Pokrycie dotyczy zarówno manualnych jak i automatycznych testów, np. 50% kodu może mieć pokrycie w testach automatycznych i jednocześnie dodatkowe 30% w testach manualnych. Możemy pokrycie testami odnieść do konkretnych funkcjonalności aplikacji. Wyższe pokrycie oznacza wyższą jakość aplikacji.

Test plans

TC - przypadek testowy

TS - zestaw testów

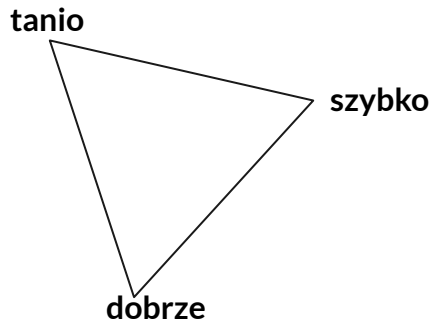




Kiedy należy zakończyć testowanie?

To jest bardzo dobre pytanie!

Istnieją trzy wyznaczniki tworzenia oprogramowania, tj. oprogramowanie można zrobić szybko, tanio albo dobrze. Nie należy znacząco skupiać na tylko jednym z nich. Najczęściej wybierane są dwa wyznaczniki. Najlepsze wyniki osiągamy jednak będąc dokładnie po środku. Jeżeli chcemy skupiać się na jednym z wyznaczników, niech to będzie jakość.



Rodzaje testów



Rodzaje testów

Istnieje wiele typów testów. Poniżej przedstawione zostały niektóre (angielskie nazwy).

- Unit testing
- Integration testing
- System testing
- Sanity testing
- Smoke testing
- Interface testing
- Regression testing
- Alpha testing
- Beta testing
- Acceptance testing
- Security testing
- Performance Testing
- Load testing
- Stress testing
- Functional testing
- Boundary value testing
- Exploratory testing
- Happy path testing
- Monkey testing
- Grey testing
- Browser compatibility testing



Testy czarnoskrzynkowe (ang. black box tests)

Testy czarnej skrzynki działają na takiej zasadzie jak nazwa wskazuje, tzn. aplikacja traktowana jest jak czarna skrzynka, czyli nie znamy kodu aplikacji. Z tego powodu należy przygotować odpowiednio dane wejściowe. Dane wyjściowe otrzymane z aplikacji porównywane są z oczekiwanym wyjściem. Inna nazwa dla testów czarnej skrzynki to testy funkcjonalne, ponieważ dotyczą funkcjonalności aplikacji.





Testy białoskrzynkowe (ang. white box tests)

Testy białej skrzynki różnią się od czarnoskrzynkowych tym, że w przypadku testów białej skrzynki znamy kod aplikacji. W tym przypadku testy automatyczne tworzone są przez programistów (zgodnie z ISTQB). Należy w testach jednostkowych przetestować klasy, metody, funkcje oraz przekazywanych parametrów.



Testy szarej skrzynki (ang. grey box testing)

Testy szarej skrzynki są kombinacją testów czarnej i białej skrzynki. W tym przypadku testujemy aplikację tak za pomocą testów funkcjonalnych mając jednocześnie wiedzę na temat kodu aplikacji.



Wydajnościowe, obciążeniowe i stres testy

(ang. performance, load and stress tests)

Często obciążeniowe i stres testy są często klasyfikowane jako wydajnościowe. Nie jest to jednak prawdą i każdy typ powinien być rozpatrywany jako osobna kategoria.

Testy wydajnościowy mają na celu znalezienie słabych stron aplikacji. Metrykami wykorzystywanymi są m.in. za czas odpowiedzi, skalowalność czy zużycie zasobów. Celem jest znalezienie miejsc, które należy poprawić pod kątem wydajności.

Testy obciążeniowe mają za zadanie sprawdzenie jak zachowa się aplikacja przy maksymalnym obciążeniu. Najczęściej testy takie trwają wiele godzin czy dni, ponieważ niektóre błędy można wykryć przy dłuższym wykorzystaniu aplikacji, np. wycieki pamięci.

Stres testy mają na celu sprawdzenie jak zachowa się aplikacja przy znacznie większym od maksymalnego obciążenia. Celem jest sprawdzenie zachowania aplikacji w przypadku nagłego skoku obciążenia aplikacji, np. sklep internetowy w okresie świątecznym.



Testy regresyjne (ang. regression tests)

Testy regresyjne lub regresywne to takie testy, które są wykonywane zawsze gdy wprowadzamy zmiany w funkcjonalności aplikacji. Najczęściej wszystkie zestawy testów są wykonywane po każdej zmianie, włączając automatyczne jak i manualne testy. Celem testów regresyjnych jest znalezienie błędów w istniejących funkcjonalnościach, które działały poprawnie do tej pory. Chodzi o sprawdzenie czy zmiany nie spowodowały pojawienie się nowych błędów w działających funkcjonalnościach.

Bugi



Przyjaciel Bug

Nie wykryte błędy są niebezpieczne, ponieważ często są tykającymi bombami. Nigdy nie wiadomo, kiedy takie błędy spowodują znaczne straty. Nigdy nie wiadomo ile takich błędów mamy w kodzie, ale staramy się doprowadzić do jak najmniejszej ich liczby. Dlatego więcej wykrytych błędów wcale nie musi znaczyć gorszej jakości oprogramowania.

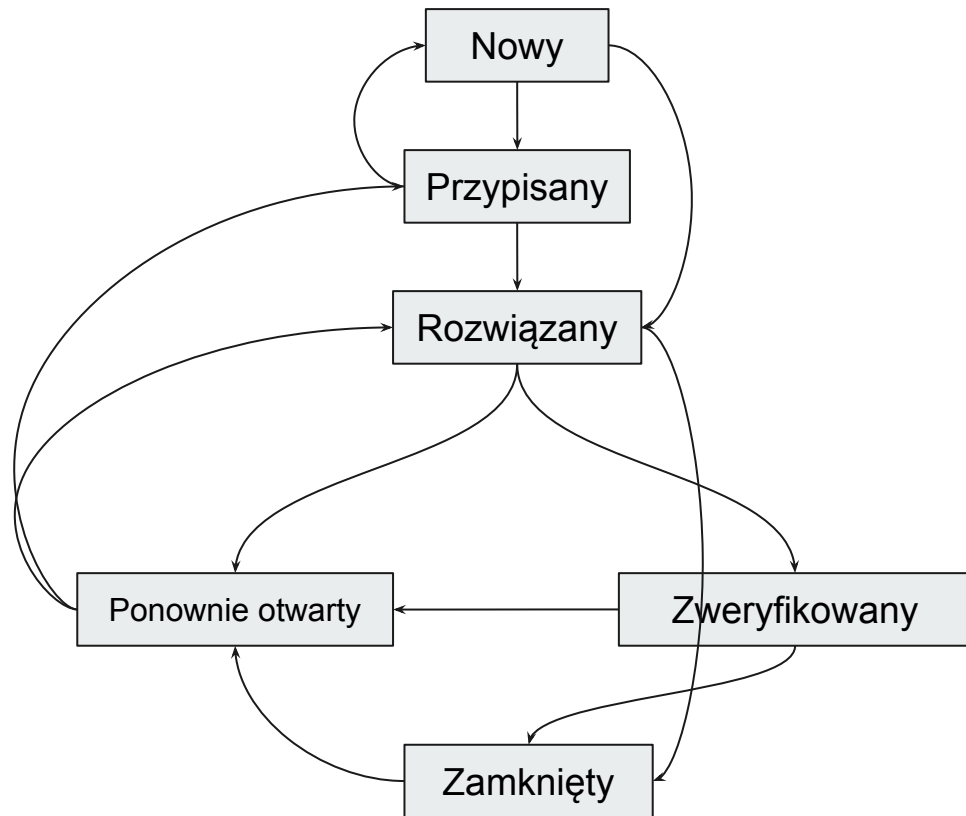
Liczba błędów może być metryką mówiącą o jakości oprogramowania. Wszystko zależy od ilości ich wykrycia i częstotliwości.

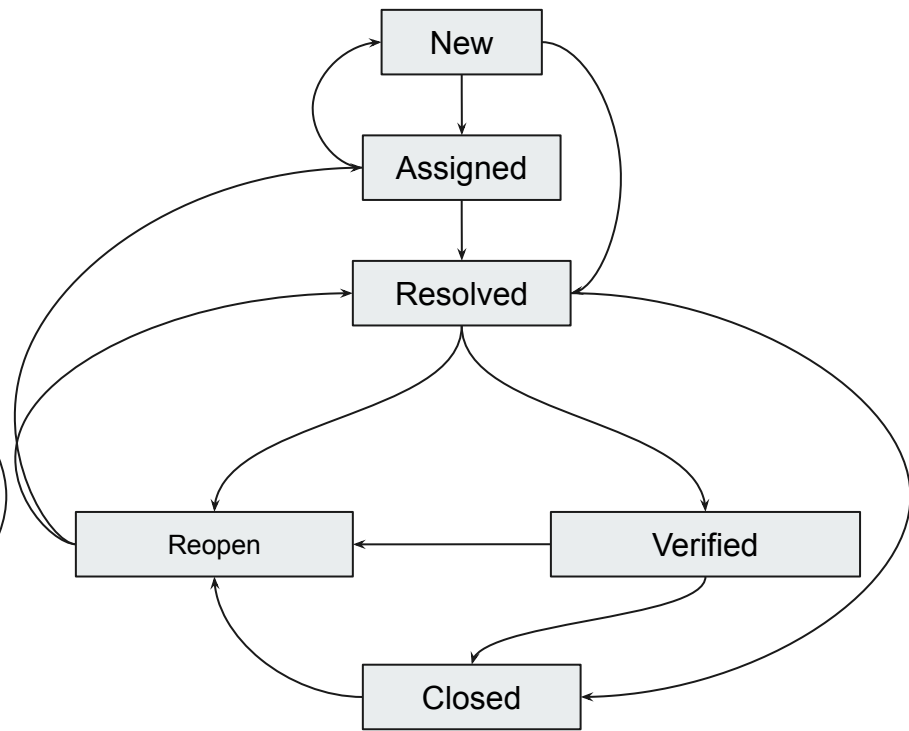
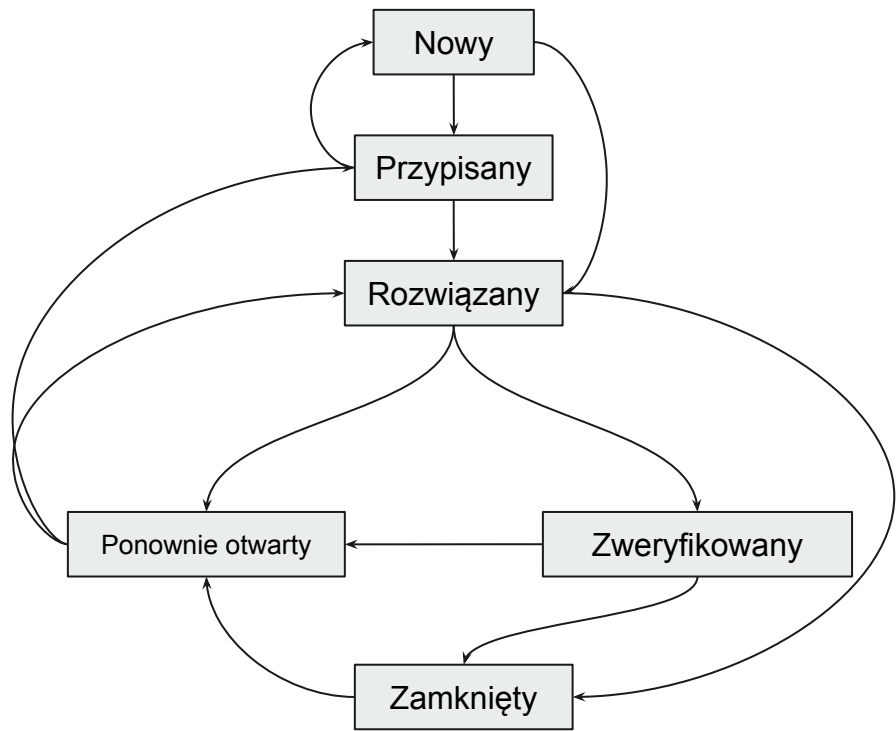
Życie bugów

(ang. Bug lifecycle)

Staramy się zawsze doprowadzić do zamknięcia buga.

Każdy błąd może zostać potraktowany w innych sposób. Rozwiązany może być jako: duplikat (ang. **duplicated**), poprawiony (ang. **fixed**), nieprawidłowy (ang. **invalid**), przeniesiony (ang. **moved**), nierozwiązany (ang. **won't fix**) oraz działa (ang. **works for me**).







Raportowanie *bugów*

Tytuł

Priority

Moduł

Dotkliwość

Przypisany

Status

Kroki

1.
2.
3.
4.

Oczekiwany wynik

Uzyskany wynik

Metodologie



Podejścia w tworzeniu oprogramowania

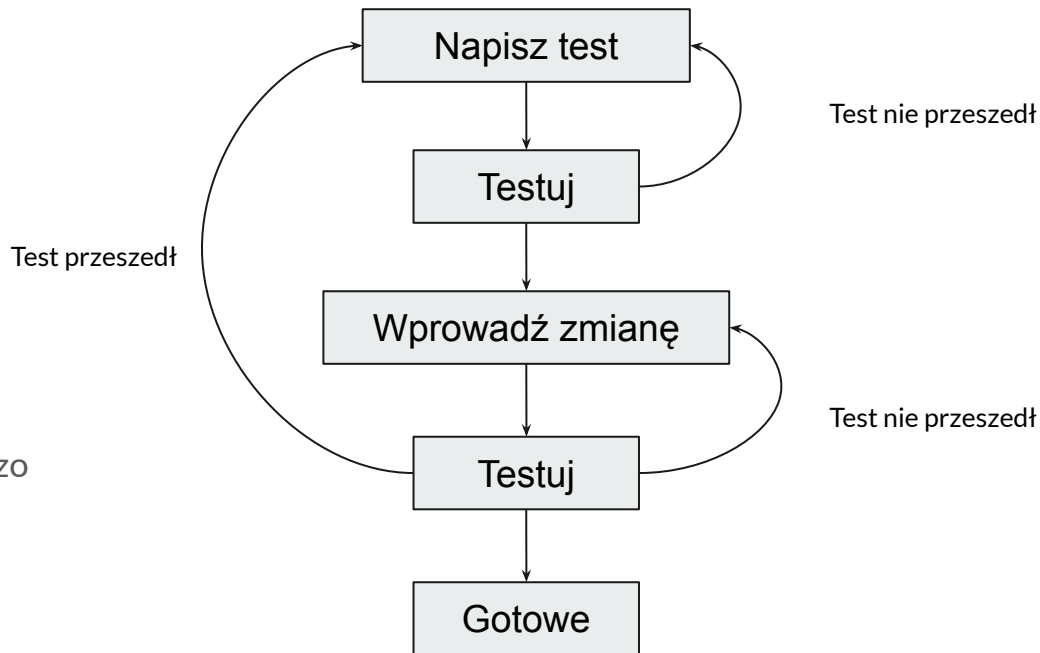
Istnieje wiele podejść, niektóre podane zostały poniżej w j. angielskim:

- acceptance test-driven development (ATDD),
- behavior-driven development (BDD),
- cross-functional team,
- continuous integration (CI),
- domain-driven design (DDD),
- iterative and incremental
- pair programming,
- planning poker,
- refactoring,
- scrum events (sprint planning, daily scrum, sprint review and retrospective),
- test-driven development (TDD),
- agile testing,
- timeboxing,
- user story,
- story-driven modeling,
- retrospective,
- velocity tracking,
- user story mapping.

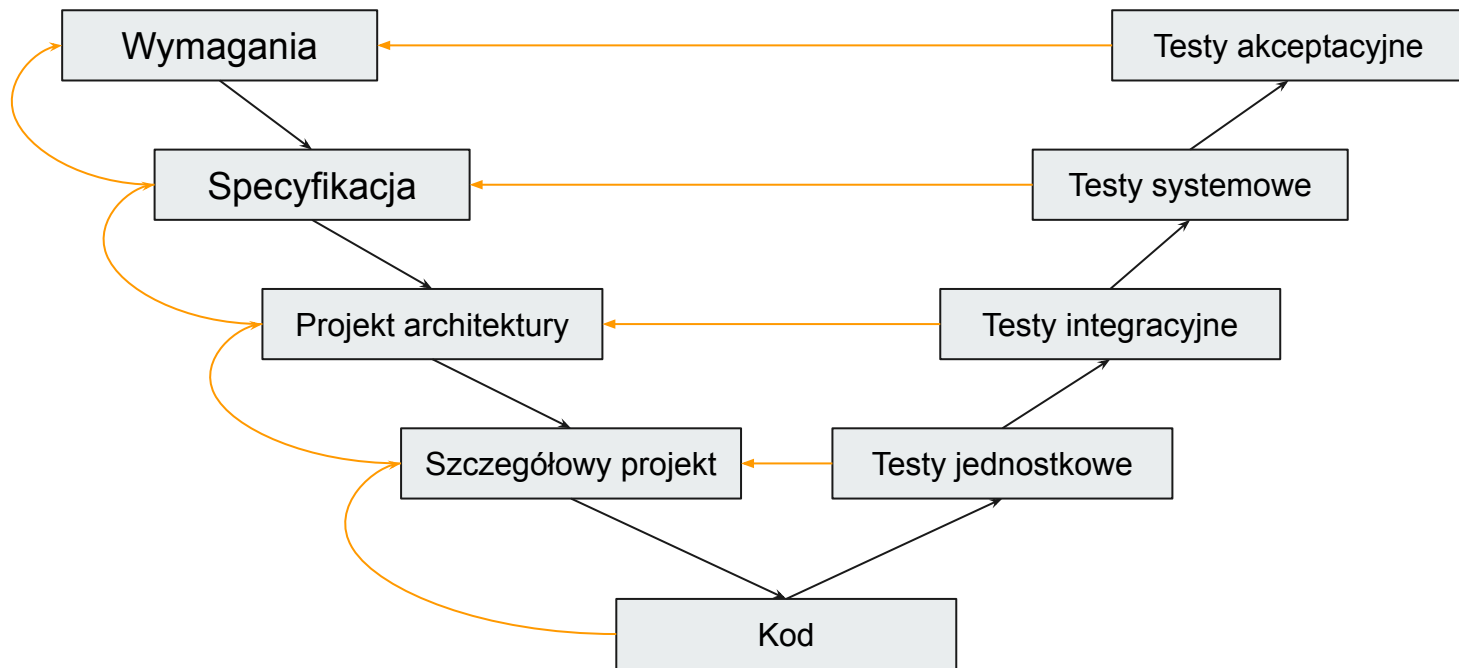
TDD

W podejściu TDD testy pisane są przed implementacją funkcjonalności.

Zaletą tego podejścia jest pokrycie funkcjonalności w testach, ale jest bardzo trudne we wdrożeniu.



V-model

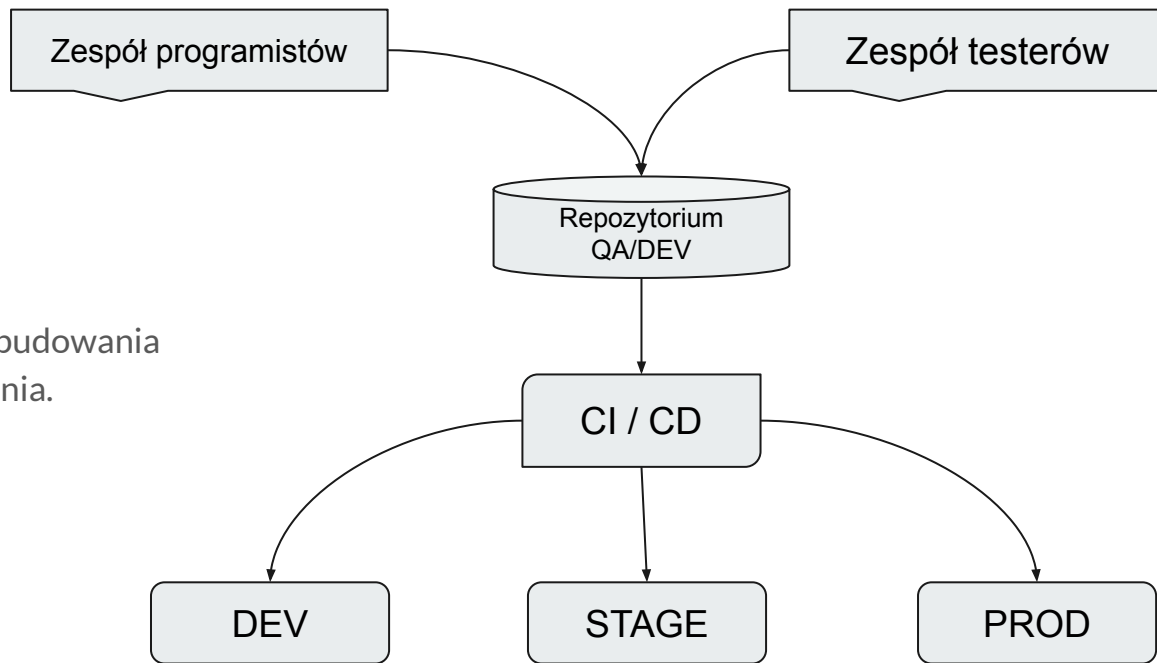


Continuous integration

Mamy trzy środowiska:

- development,
- staging,
- production.

Continuous integration służy do budowania aplikacji, testowania oraz wdrażania.





Continuous delivery

Różnica pomiędzy CI a CD jest taka, że w tym drugim przypadku, aplikacja jest budowana, testowana oraz wdrażana przy każdym wprowadzeniu zmian w kodzie.

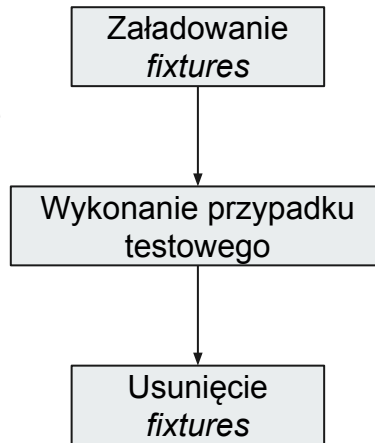
Dodatkowe terminy związane z jakością



Fixtures

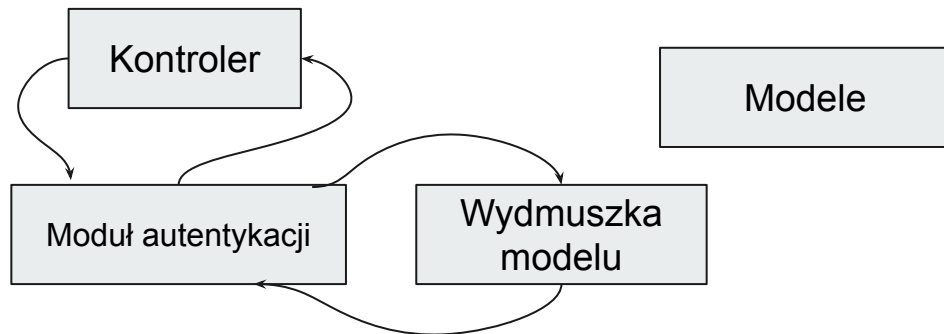
Fixtures to dane przygotowane na potrzeby testów. Bardzo często do przetestowania funkcjonalności wymagane są dane, np. dane do logowania czy dane, które należy pobrać za pomocą aplikacji. Aby test był możliwy do przeprowadzenia, należy najpierw wypełnić bazę odpowiednimi danymi, które są ładowane i znane są jako fixtures. Często przechowywane są w formacie XML, YAML, JSON czy SQL.

Bazda powinna zostać wyczyszczona po wykonaniu każdego przypadku testowego.



Wydmuszka (ang. mock)

Nie zawsze możemy lub chcemy przetestować całą aplikację. Niektóre funkcjonalności trzeba przetestować osobno. Trudno jednak przetestować niektóre funkcjonalności nie angażując w testy inne. Przykładem może być autentykacja. Możemy jednak stworzyć model użytkownika za pomocą wydmuszki (mock), który symuluje działanie modelu użytkownika. To samo możemy zrobić z niemal każdą klasą, serwisem czy biblioteką.





Wzorzec POM (Page Object Model)

Page Object Model jest wzorcem bardzo popularnym w tworzeniu testów automatycznych. Jedną z zalet tego wzorca jest łatwa utrzymywalność kodu napisanego zgodnie ze wzorcem POM. POMem jest każda strona aplikacji, tj. strona reprezentowana jest przez klasę. Dzięki temu wraz ze zmianą elementów na stronie, należy jedynie zmienić kod w danej klasie.

Notyfikacje



Manifest

```
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
```




Weryfikacja

areNotificationsEnabled()



Zależności

```
val core_version = "1.6.0"  
dependencies {  
    implementation("androidx.core:core-ktx:$core_version")  
}
```



Tworzenie notyfikacji

```
var builder = NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle(textTitle)
    .setContentText(textContent)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
```



Czcionki

```
.setStyle(NotificationCompat.BigTextStyle()  
    .bigText("Much longer text that cannot fit one line..."))
```



Serwis

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
```

```
<service android:name=".PromocjeService" android:exported="false" />
```



Serwis

```
class PromocjeService : Service() {  
  
    private val binder: Binder = PromocjeBinder()  
  
    override fun onBind(p0: Intent?): IBinder = binder  
  
    inner class PromocjeBinder : Binder() {  
        val service: PromocjeService  
        get() = this@PromocjeService  
    }  
}
```



Serwis

```
private lateinit var promocjeService: PromocjeService
private var isPromocjeServiceBound: Boolean = false
private val promocjeServiceConnection = object : ServiceConnection {

    override fun onServiceConnected(name: ComponentName?, binder: IBinder?) {
        promocjeService = (binder as PromocjeService.PromocjeBinder).service
        isPromocjeServiceBound = true
    }

    override fun onServiceDisconnected(p0: ComponentName?) {
        isPromocjeServiceBound = false
    }
}
```



Serwis

```
val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager  
notificationManager.createNotificationChannel(channel)
```




Serwis

```
fun createNotification(  
    context: Context,  
    title: String,  
    content: String,  
) {  
    val builder = NotificationCompat.Builder(context, CHANNEL_ID)  
    val notification = builder.setContentTitle(title).setContentText(content)  
        .setSmallIcon(R.drawable.ic_launcher_foreground).build()  
    NotificationManagerCompat.from(context).notify(id, notification)  
}
```



Bubble

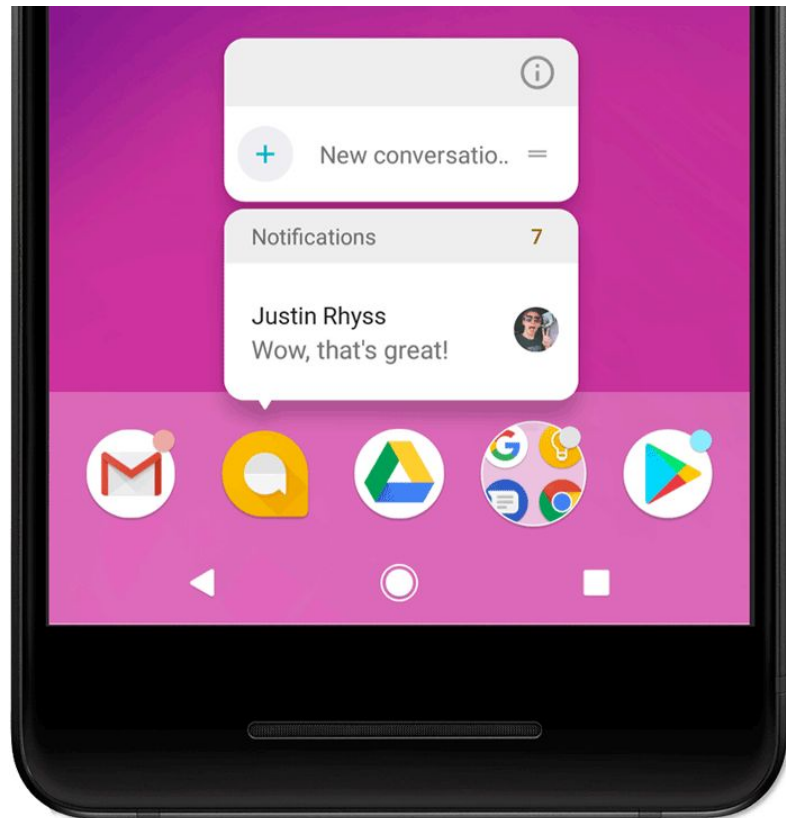
```
val bubbleData = Notification.BubbleMetadata.Builder(bubbleIntent,  
    Icon.createWithResource(context, R.drawable.icon))  
    .setDesiredHeight(600)  
    .build()  
  
.setBubbleMetadata(bubbleData)
```

Badge

```
.setShowBadge (false)
```

```
.setNumber (messageCount)
```

```
.setBadgeIconType (NotificationCompat.BADGE_ICON_SMALL)
```





Akcje

```
val snoozeIntent = Intent(this, MyBroadcastReceiver::class.java).apply {  
    action = ACTION_SNOOZE  
    putExtra(EXTRA_NOTIFICATION_ID, 0)  
}
```

```
val snoozePendingIntent: PendingIntent =  
    PendingIntent.getBroadcast(this, 0, snoozeIntent, 0)  
val builder = NotificationCompat.Builder(this, CHANNEL_ID)  
    .setSmallIcon(R.drawable.notification_icon)  
    .setContentTitle("My notification")  
    .setContentText("Hello World!")  
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)  
    .setContentIntent(pendingIntent)  
    .addAction(R.drawable.ic_snooze, getString(R.string.snooze),  
        snoozePendingIntent)
```



Odpowiedzi

```
private val KEY_TEXT_REPLY = "key_text_reply"
var replyLabel: String = resources.getString(R.string.reply_label)
var remoteInput: RemoteInput = RemoteInput.Builder(KEY_TEXT_REPLY).run {
    setLabel(replyLabel)
    build()
}
```

```
var replyPendingIntent: PendingIntent =
    PendingIntent.getBroadcast(applicationContext,
        conversation.getConversationId(),
        getMessageReplyIntent(conversation.getConversationId()),
        PendingIntent.FLAG_UPDATE_CURRENT)
```



Odpowiedzi

```
var action: NotificationCompat.Action =  
    NotificationCompat.Action.Builder(R.drawable.ic_reply_icon,  
        getString(R.string.label), replyPendingIntent)  
        .addRemoteInput(remoteInput)  
        .build()  
  
.addAction(action)
```



Odpowiedzi

```
private fun getMessageText(intent: Intent): CharSequence? {  
    return RemoteInput.getResultsFromIntent(intent)?.getCharSequence(KEY_TEXT_REPLY)  
}
```

Grupy

```
.setGroup(ID_GRPY)
```

```
.setGroupSummary(true)
```

 Gmail • 4m ▾

Justin Rhyss It's Friday! Let's start making plans for the we...
Ali Connors Game tomorrow Don't forget to bring your... + 1

Expanded

 Gmail • 4m ▲

4m ▾

Justin Rhyss
It's Friday! Let's start making plans for the weeken...



12m ▾

Ali Connors
Game tomorrow Don't forget to bring your jersey si...

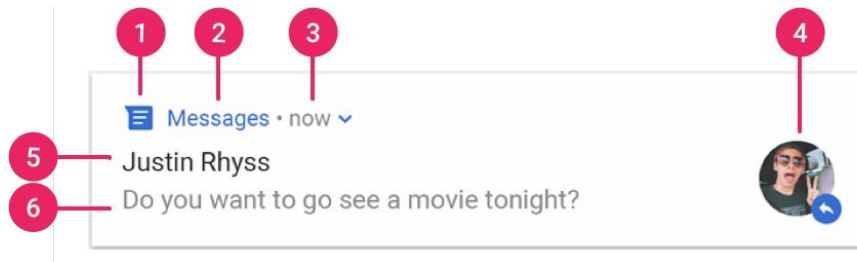


23m ▾

Mary Johnson
How did it go this week? Are you going to be in for...



Wygląd



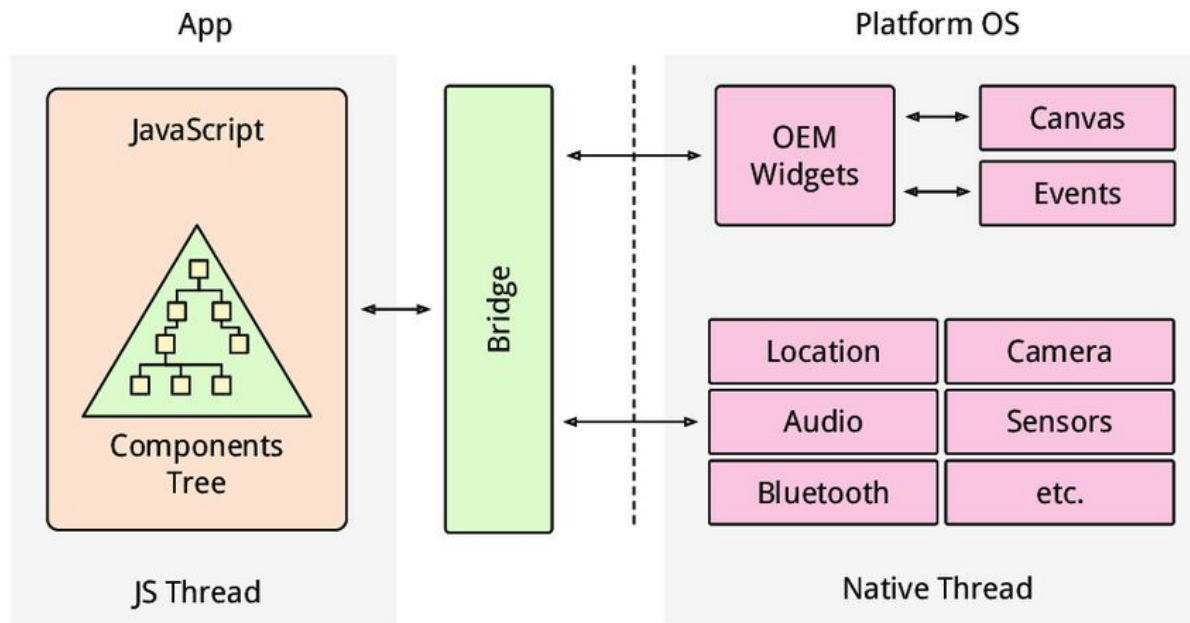


Bezpieczeństwo

```
.setAuthenticationRequired(true)
```

React Native

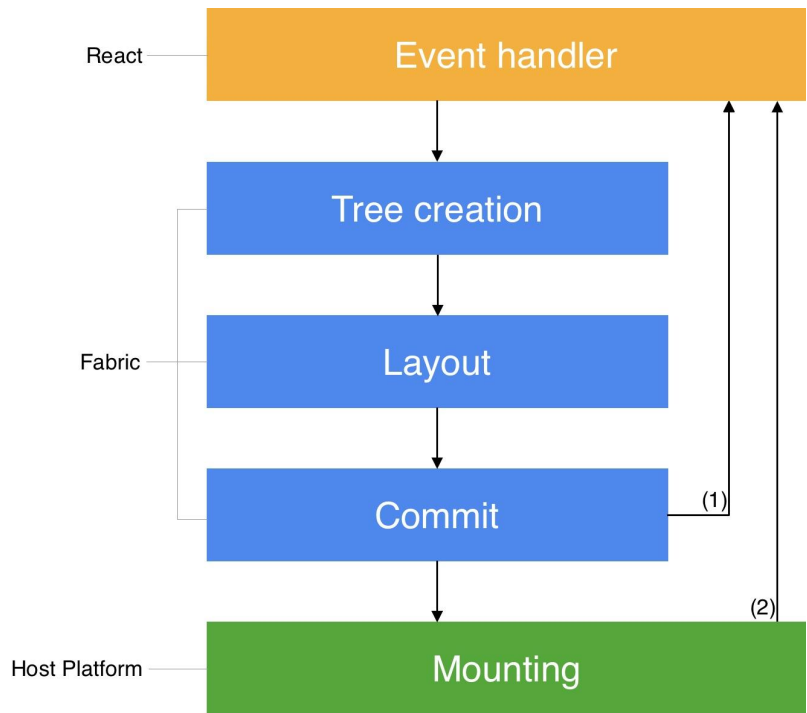
Architektura



Budowanie

Metro <https://facebook.github.io/metro/>

Fabric



1. Events from core. For example: onLayout
2. Events from host platform. For example: onChange



Budowanie

```
export ANDROID_SDK_ROOT=$HOME/Library/Android/sdk
```

```
export PATH=$PATH:$ANDROID_SDK_ROOT/emulator
```

```
export PATH=$PATH:$ANDROID_SDK_ROOT/platform-tools
```



Budowanie

```
npm run android
```

```
npm run web
```



Komponenty

ImageView

Events

Fragment



UI

<View>

<ViewGroup>

<Text>

<TextView>

<Image>

<ImageView>

<ScrollView>

<ScrollView>

<TextInput>

<EditText>

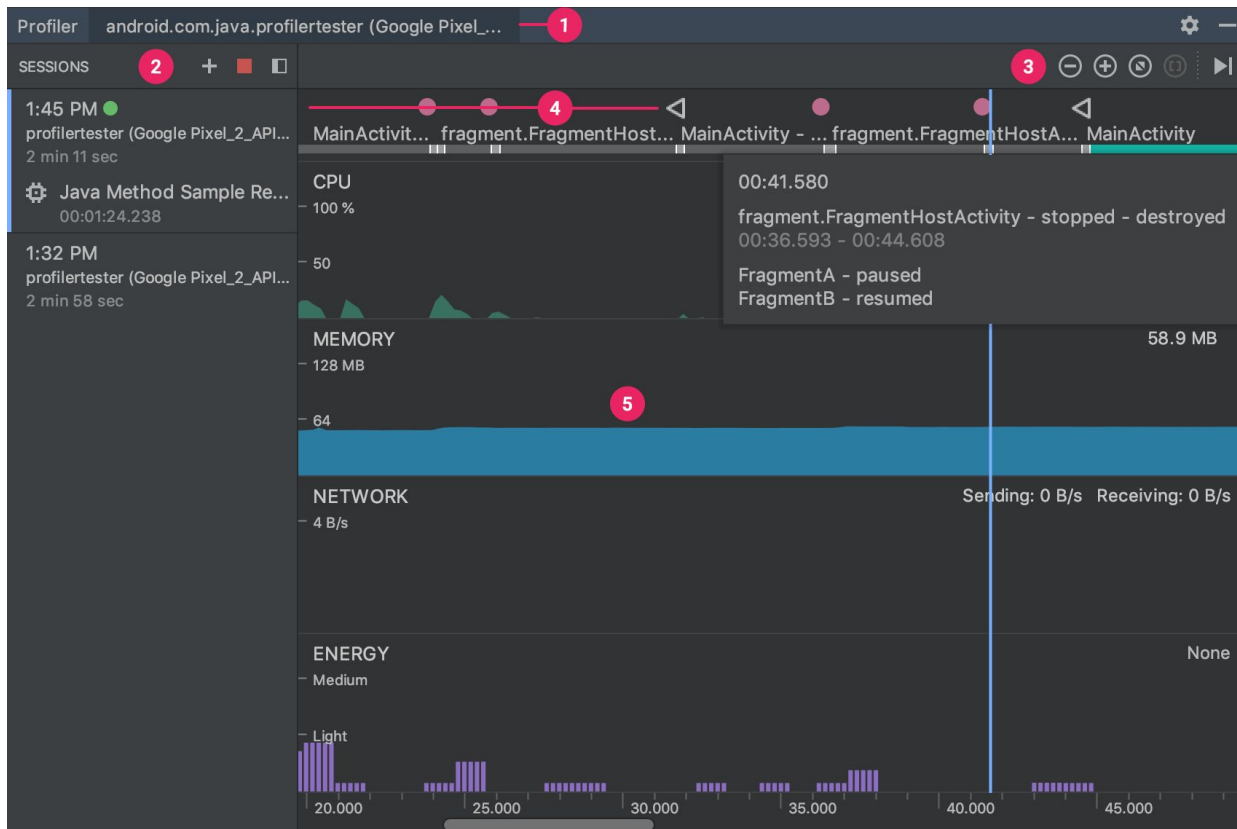
UI

```
import React from 'react';
import { View, Text, Image, ScrollView, TextInput } from 'react-native';

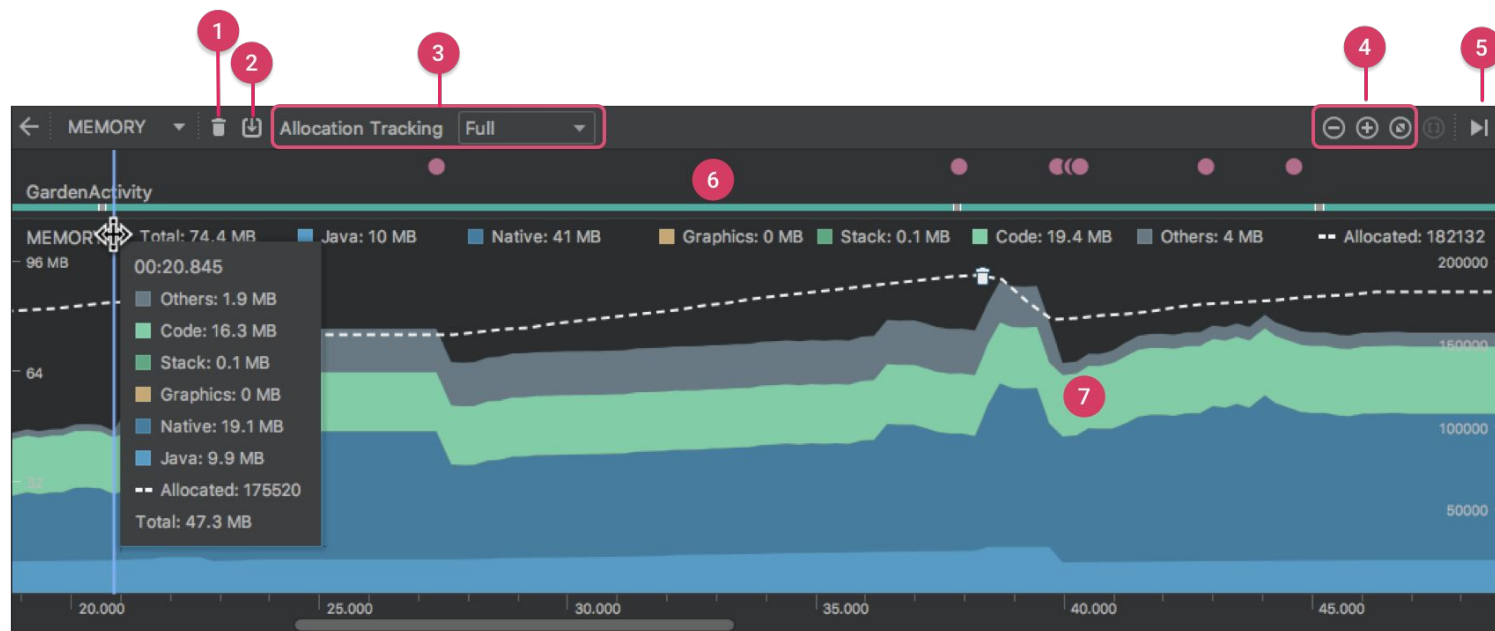
const App = () => {
  return (
    <ScrollView>
      <Text>Some text</Text>
      <View>
        <Text>Some more text</Text>
        <Image
          source={{
            uri: 'https://reactnative.dev/docs/assets/p_cat2.png' ,
          }}
          style={{ width: 200, height: 200 }}
        />
      </View>
      <TextInput
        style={{
          height: 40,
          borderColor: 'gray',
          borderWidth: 1
        }}
        defaultValue="You can type in me"
      />
    </ScrollView>
  );
};
```

Performance

Profiler



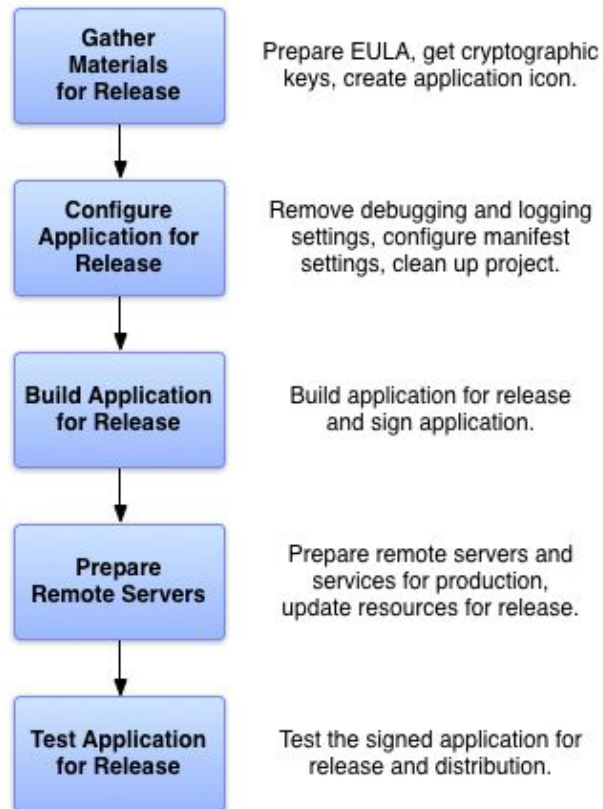
Profiler



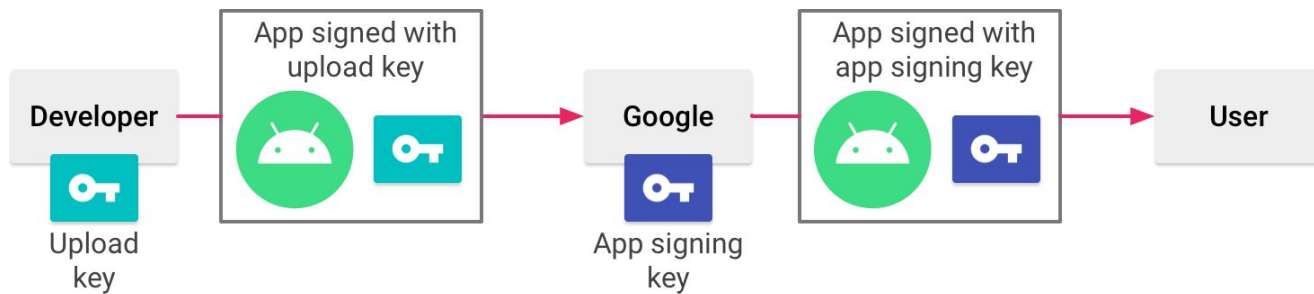
Google Play



Publikowanie



Publikowanie





Bibliografia

[1] Dawn Griffiths, David Griffiths, Head First Android Development, 3rd Edition, O'Reilly 2021

[2] Elton Stoneman, Learn Docker in a Month of Lunches, Manning 2020

[3] Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Pearson 2008

[4] Alexey Soshin, Hands-On Design Patterns with Kotlin, Packt 2018

[5] Alex Forrester, Eran Boudjnah, Alexandru Dumbravan, Jomar Tigcal, How to Build Android Apps with Kotlin, Packt 2021

[6] ISTQB Foundation Level syllabus:
<https://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html>