APLIKACJE MOBILNE

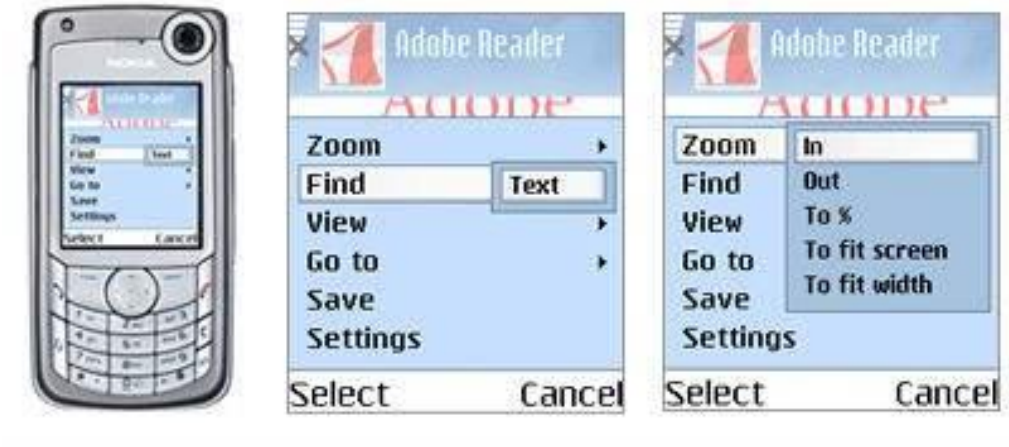
Wykład

dr Artur Bartoszewski

Przegląd systemów dla urządzeń mobilnych

Aplikacje mobilne

NIEROZWIJANE



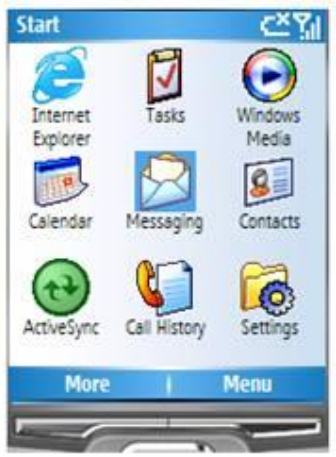
symbian OS

- ✓ Wywodzi się z systemu EPOC dla palmtopów firmy PSION
- ✓ Psion, Nokia, Ericssoni Motorola założyli w 1999 r. firmę Symbian
- ✓ Pierwszy Symbian 6.0 w 2000 roku (numerek odziedziczył po EPOCu)
- ✓ W 2008 Nokia przejęła większość udziałów i przekształciła firmę w fundacje Symbian
- ✓ Podstawowe języki programowania:
  - do 2010 r: Symbian C++
  - od 2010: standardowy C++ wraz z Qt
  - można używać też innych języków (Python, Java ME, .NET…)

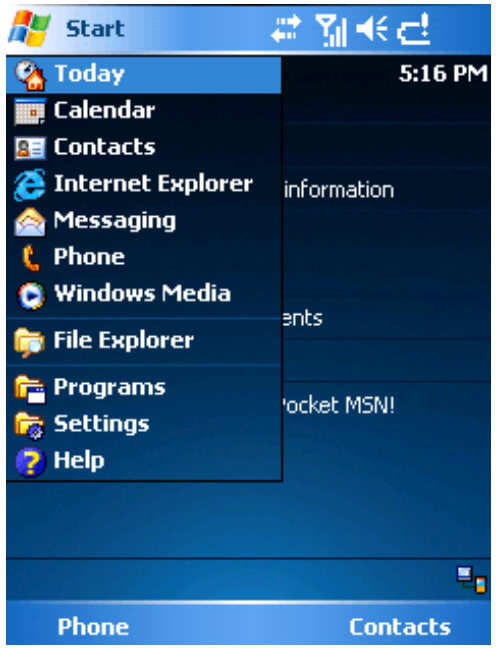
Aplikacje mobilne

NIEROZWIJANE

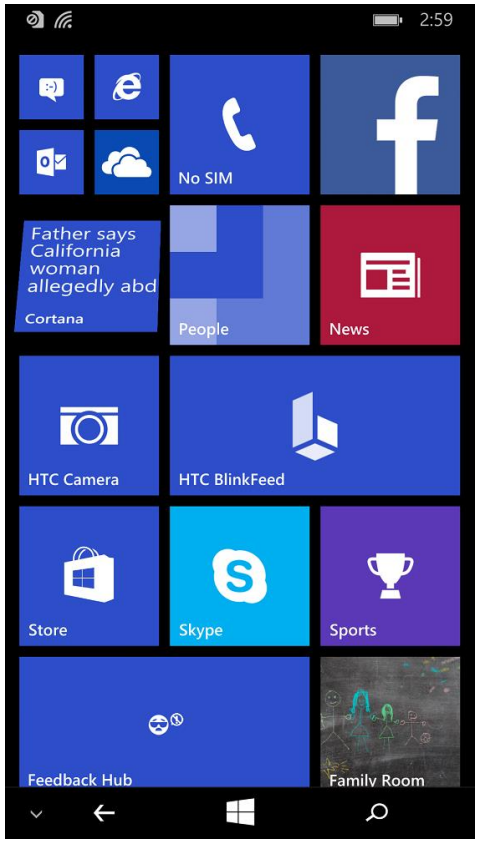
Windows CE



Windows Mobile



Windows Phone



- ✓ Protoplasci: Windows CE (1996),
- ✓ PocketPC 2000-2002
- ✓ Windows Mobile 2003,
- ✓ Windows Mobile 5 (2005), 6 (2007), 6.5 (2010)
- ✓ Wersja dla smartfonów: Windows Phone7 (2010) –niekompatybilny z wcześniejszymi używa interfejsu METRO
- ✓ Języki programowania: C#, VB Sliverlight XNA Wgrany pakiet Office

iOS

- ✓ Wywodzi się z systemu Mac Os X, do 2010 r. iPhone OS Oryginalnie powstał dla iPhone'a, później rozszerzony do obsługi iPoda, iPadai AppleTV
- ✓ Zamknięty system, brak możliwości instalacji aplikacji spoza AppStore
- ✓ Dość kosztowna możliwość dystrybucji aplikacji
- ✓ Język programowania: Objective-C Wymagany komputer Mac z procesorem Intela
- ✓ Prosty, czytelny, bardzo sprawnie działający interfejs użytkownika

Aplikacje mobilne

Aplikacje mobilne

Android

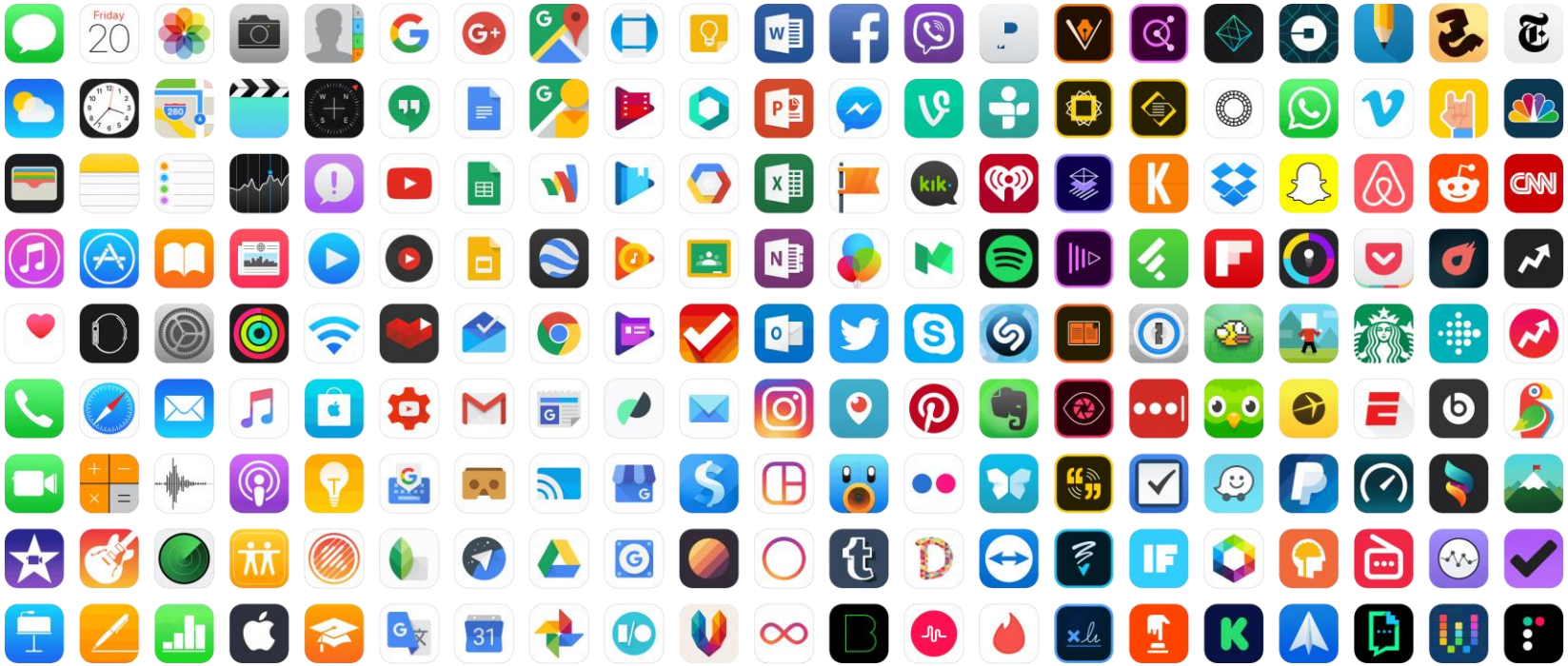
- ✓ W lipcu 2005 roku Google kupuje firmę Android Inc., produkującą oprogramowanie dla urządzeń mobilnych
- ✓ Premiera systemu i pierwsze smartfony: połowa 2008
- ✓ Oparty na Linuksie, z własną maszyną wirtualną Dalvik / ART
- ✓ Język programowania: Java, z możliwością wstawek w C++
- ✓ System otwarty, umożliwia instalowanie aplikacji spoza Marketu
- ✓ Dostępne źródła systemu

Aplikacje mobilne

ANDROID PLATFORM VERSION		API LEVEL	CUMULATIVE DISTRIBUTION
4.4	KitKat	19	
5	Lollipop	21	99,6%
5.1	Lollipop	22	99,4%
6	Marshmallow	23	98,2%
7	Nougat	24	96,3%
7.1	Nougat	25	95,0%
8	Oreo	26	93,7%
8.1	Oreo	27	91,8%
9	Pie	28	86,4%
10	Q	29	75,9%
11	R	30	59,8%
12	S	31	38,2%
13	T	33	22,4%

Last updated: October 1, 2023

Rodzaje aplikacji



Aplikacje mobilne

Dla aplikacji dedykowanych na urządzenia mobilne wyróżnić można cztery podstawowe metody implementacji:

- aplikacje natywne, czyli przeznaczone tylko i wyłącznie na jedną platformę;
- aplikacje wieloplatformowe, przeznaczone do użytku w więcej niż jednym systemie operacyjnym;
- aplikacje PWA (*ang. Progressive Web Applications*), działające poprzez silnik przeglądarki internetowej,
- aplikacje hybrydowe, będące połączeniem aplikacji natywnych oraz aplikacji PWA.

Aplikacje mobilne

Aplikacje natywne (ang. native applications) tworzone są z myślą o jednym konkretnym systemie operacyjnym.

- Są w pełni zoptymalizowane pod kątem docelowego systemu, dzięki czemu, mogą zaoferować najlepszą wydajność i dostęp do wszystkich funkcji urządzenia.
- Tworzone są przy użyciu specyficznych narzędzi i technologii dedykowanych konkretnej platformie.
- Wymaganie jest stosowanie odpowiednich języków programowania, środowisk programistycznych, zestawów SDK (ang. Software Development Kit) oraz dedykowanych narzędzi programistycznych.

Platforma	Języki programowania	Środowiska programistyczne
Android	Java, Kotlin	Android Studio, Eclipse, Visual Studio
iOS	C (obiektowy), Swift	Xcode
Windows Phone	C#	Visual Studio, Visual Studio Code

Aplikacje mobilne

Aplikacje wieloplatformowe (ang. cross-platforms applications) to programy, które wykorzystują składowe platform natywnych, mogą jednak być kompilowane na wielu systemach operacyjnych.

Efekt ten osiągnięto dodając nową warstwę abstrakcji pomiędzy SDK, czyli środowiskiem z jakiego korzysta programista a natywnym API sytemu operacyjnego.

Jej zadaniem jest tłumaczenie wygenerowanego za pomocą SDK kodu (wspólnego dla wszystkich platform) na natywny kod docelowego urządzenia.

Technologia	Obsługiwane platformy mobilne	Języki programowania	Środowiska programistyczne
Xamarin	Android, iOS	C#	Visual Studio
React Native	Android, iOS	JavaScript	Visual Studio Code
Unity	Android, iOS	C#	Visual Studio,Visual Studio Code
Flutter	Android, iOS, platformy sieciowe	Dart	Visual Studio Code,Android Studio

Aplikacje mobilne

Aplikacje PWA (ang. Progressive Web Applications) to rodzaj aplikacji internetowych, które łączą cechy stron internetowych i tradycyjnych aplikacji mobilnych.

Opierają one swoje działanie na silniku przeglądarki internetowej udostępnianym przez API każdego systemu operacyjnego przeznaczonego dla urządzeń mobilnych.

Mówiąc prościej, aplikacja PWA to strona internetowa wyświetlana przez aplikację dysponującą pełnymi możliwościami przeglądarki internetowej lecz pozbawioną jej interfejsu sterującego, czyli zakładek, paska adresu, przycisku wstecz itp. Dzięki temu odpowiednio przygotowana aplikacja PWA udawać może aplikację natywną.

Technologia	Obsługiwane platformy mobilne	Języki programowania	Środowiska programistyczne
IONIC	Android, iOS,	HTML+CSS+JavaScript	Visual Studio Code, Atom
Polymer	Platformy sieciowe	HTML+CSS+JavaScript	Atom
AngularJs	Android, iOS, platformy sieciowe	HTML+CSS+JavaScript	Visual Studio Code, Atom, Angular IDE
VueJS	Android, iOS, platformy sieciowe	HTML+CSS+JavaScript	Visual Studio Code, Atom, WebStorm

Aplikacje mobilne

Aplikacje hybrydowe (ang. hybrid applications), znane również jako hybrydowe aplikacje mobilne, stanowią połączenie cech charakterystycznych zarówno dla aplikacji natywnych, jak i aplikacji sieciowych.

Struktura tych aplikacji opiera się na komponentach tworzonych przy użyciu narzędzi webowych, przy jednoczesnym wykorzystaniu zewnętrznej otoki w postaci aplikacji natywnej do zapewnienia niezbędnej funkcjonalności.

Technologia	Obsługiwane platformy mobilne	Języki programowania	Środowiska programistyczne
IONIC	Android, iOS, aplikacje PWA	HTML+CSS+JavaScript	Visual Studio Code, Atom, ALM
Xamarin	Android, iOS	C#	Visual Studio
Flutter	Android, iOS, platformy sieciowe	Dart	Visual Studio Code, Android Studio

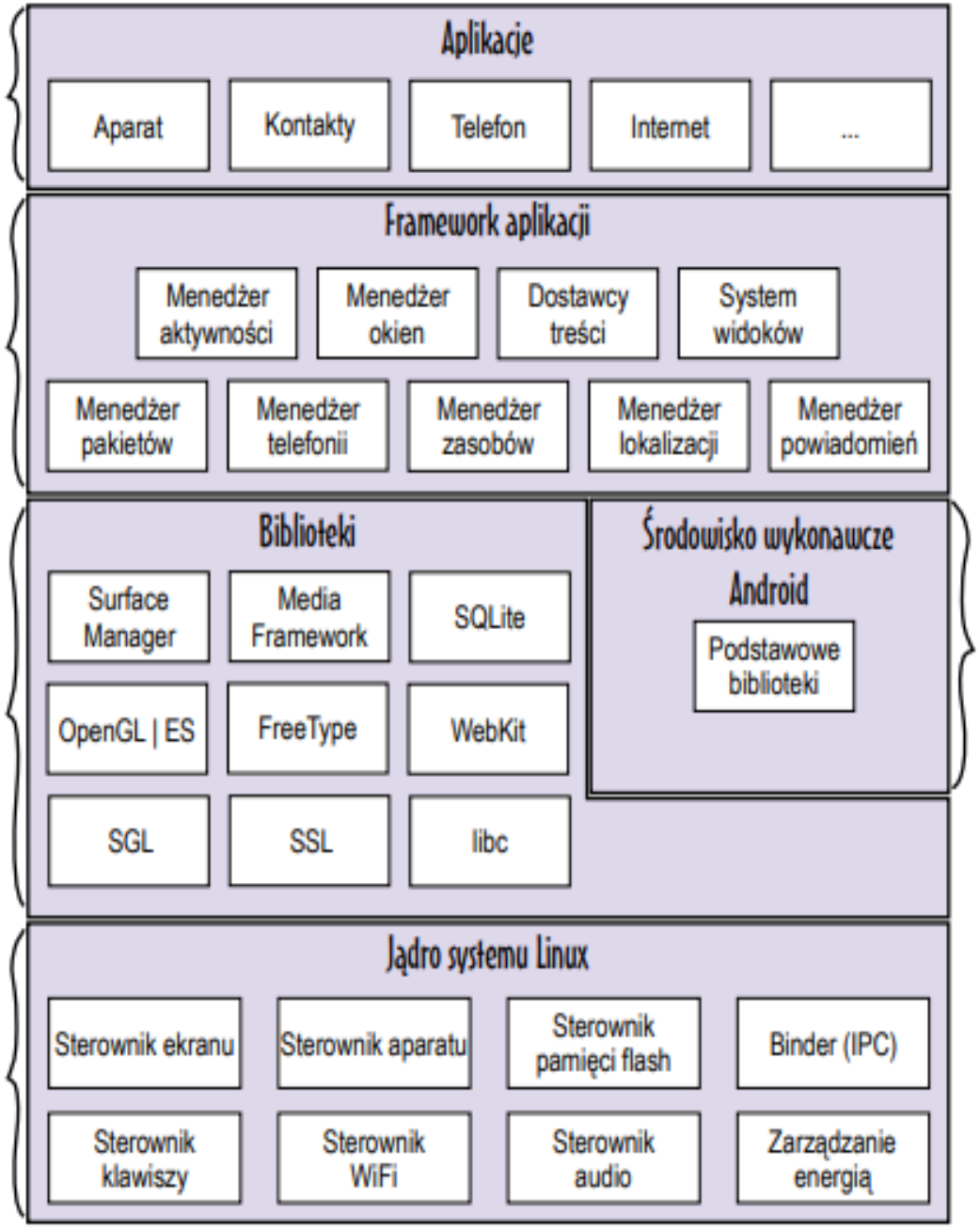
Budowa systemu Android

Android jest wyposażony w zestaw podstawowych aplikacji, takich jak Kontakty, Kalendarz i Mapy, oraz w przeglądarkę WWW.

Pisząc własne aplikacje, mamy dostęp do tych samych podstawowych API, które są używane przez podstawowe aplikacje. Używając tych API, kontrolujemy wygląd aplikacji i jej działanie.

Poniżej frameworku aplikacji ukryty jest zestaw bibliotek C i C++. Dostęp do nich zapewniają API należące do frameworku aplikacji.

Poniżej wszystkich pozostałych warstw platformy Android umieszczone jest jądro systemu Linux. To ono odpowiada za obsługę sterowników oraz podstawowych usług, takich jak zabezpieczenia i zarządzanie pamięcią.



Środowisko uruchomieniowe Androida jest wyposażone w zestaw podstawowych bibliotek, implementujących przeważającą część języka programowania Java. Każda aplikacja na Androida jest wykonywana we własnym procesie.

Linux Kernel - (jądro linuksowe) - Android opiera się na wersji jądra 2.6 dla podstawowych usług systemowych, takich jak bezpieczeństwo, zarządzanie pamięcią, zarządzanie procesami, stos sieciowy i model sterownika. Jądro działa również jako warstwa abstrakcji pomiędzy sprzętem i resztą stosu oprogramowania.

Android Runtime - (środowisko uruchomieniowe) - Android zawiera zbiór bibliotek, które dostarczają większość funkcji dostępnych w bibliotekach podstawowych języka Java. Każda aplikacja działa we własnym procesie, z własnej instancji maszyny wirtualnej Android Runtime (ART), która zastąpiła starszą maszynę Dalvik.

Została ona napisana tak, że na urządzeniu można uruchomić wiele maszyn wirtualnych. ART. wykonuje pliki wykonywalne (.dex) skonstruowane tak, aby zużywały minimalną ilość pamięci

Aplikacje mobilne

Libraries - (biblioteki) - Android zawiera zestaw bibliotek C / C++ używanych przez różne elementy systemu. Możliwości te są udostępnione programistom poprzez Application Framework.

Application Framework - (framework aplikacji) - programiści mają pełny dostęp do tego samego API, którego używają aplikacje podstawowe systemu.

Framework oferuje:

system widoków (ang. Views), które wykorzystuje się do budowania interfejsu aplikacji;

dostawców treści (ang. ContentProviders), które pozwalają aplikacjom dostęp do danych z innych aplikacji, (np. takich jak Kontakty), lub dzielenie się swoimi danymi;

menedżera zasobów (ang. Resource Manager), umożliwiający dostęp do zasobów, takich jak, grafika i pliki;

menadżera powiadomień (ang. Notification Manager), który umożliwia wszystkim aplikacjom wyświetlanie powiadomień w pasku stanu;

menadżera aktywności (ang. ActivityManager), który zarządza cyklem życia aplikacji.

Struktura aplikacji na Androida

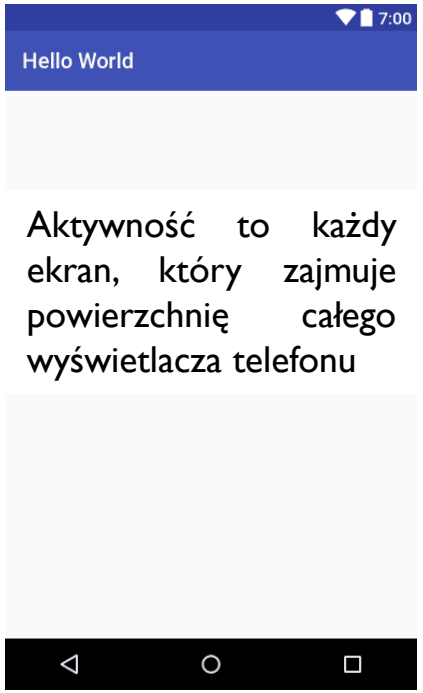
Aplikacje na system Android składają z komponentów powiązanych ze sobą w luźny sposób, komunikujących się ze sobą za pośrednictwem Intencji (Intent).

Podstawowe komponenty to:

- Aktywności (*activities)*
- Usługi (*services),*
- Dostawcy treści (*content providers)*
- Odbiorniki komunikatów (*broadcast receivers)*

Aplikacje mobilne

- ✓ Aktywności *(activities)* - *oznaczają warstwę wizualną aplikacji oraz dołączone do niej funkcjonalności.*
  - *Dana aktywność reprezentuje pojedynczy ekran zawierający interfejs użytkownika, tak więc jedna aplikacja może zawierać wiele wywołujących się nawzajem aktywności.*
  - *Aplikacje mogą również uruchamiać aktywności zawarte w innych aplikacjach – pod warunkiem, że mają odpowiednie zezwolenia.*



Widoki – (View) – to obiekty tworzące interfejs użytkownika. Za pomocą widoków użytkownik komunikuje się z aplikacją (z aktywnościami). Na widok składa się układ kontrolek zawartych w odpowiedniej kompozycji (layout).

- Budowa graficznego interfejsu użytkownika aplikacji Android opiera się na hierarchii obiektów View i ViewGroup.
- Obiekty View są zazwyczaj widocznymi elementami interfejsu, np. przyciskami i polami tekstowymi,
- obiekty ViewGroup to niewidoczne kontenery określające sposób rozmieszczenia elementów widoków potomnych, np. w siatce albo pionowej liście.

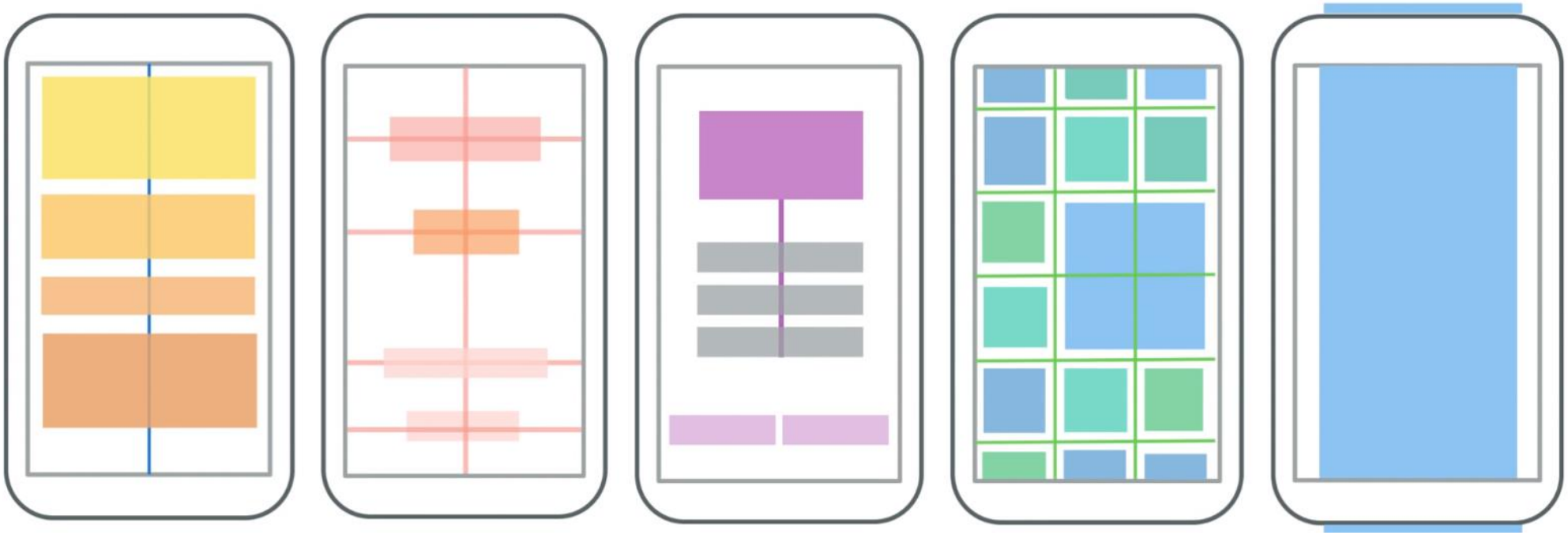
- ✓ Usługi *(services), w przeciwieństwie do aktywności, nie posiadają reprezentacji graficznej.*
  - *Działają one w tle i wykonują czasochłonne zadania lub operacje na rzecz zdalnych procesów, np. pobierają pliki z Sieci i aktualizują źródła danych, wysyłają powiadomienia, itd.*

- ✓ Intencje (*intent*) - *to rodzaj komunikatów wysyłanych do komponentów aplikacji.*
  - *Służą do wywoływania aktywności (zarówno własnego programu jak i zewnętrznych zarejestrowanych w systemie).*
  - *Aplikacja posiada w pliku manifest.xml filtr intencji – precyzujący na jakie intencje jest ona w stanie zareagować.*
  - *Intencja może użyć aplikacji która nie jest aktualnie uruchomiona – wystarczy, by była zarejestrowana w systemie.*

Aplikacje mobilne

- ✓ Dostawcy treści (*content providers*) *to komponenty służące do udostępniania i współdzielenia danych pomiędzy aplikacjami.*
  - *Pracują one w podobny sposób do relacyjnych baz danych.*
  - *Informacje są udostępniane w formie struktur przypominających tabele baz danych.*
- ✓ Odbiorniki komunikatów (*broadcast receivers*) *umożliwiają tworzenie aplikacji sterowanych zdarzeniami (event-driven applications).*
  - *Nie wyświetlają interfejsu użytkownika, ale mogą tworzyć komunikaty na pasku powiadomień, informujące np. o otrzymaniu wiadomości lub zakończeniu pobierania pliku.*
  - *Dzieje się to w skutek aktywowania przez system lub inne aplikacje.*

Layout aplikacji



Widoki

Podstawowymi elementami z których budowany jest interfejs użytkownika w systemie Android są widoki (ang. View).

Jest to klasa bazowa dla elementów, które są używane do stworzenia interaktywnych komponentów takich jak przyciski, pola tekstowe, przełączniki itp.

Widok zajmuje na ekranie prostokątny obszar i jest odpowiedzialny za narysowanie obiektu i obsługę zdarzeń.

Model pudełkowy elementu typu View.
Od modelu pudełkowego znanego z HTML i CSS różni się on brakiem możliwości definiowania obramowania. Wynika to z faktu, że w przypadku Androida mamy do czynienia z predefiniowanymi kontrolkami.

Layouty – ViewGroup

Oprócz zwykłych kontrolek istnieje specjalna podgrupa widoków, również dziedzicząca po klasie View, określanych jako ViewGroup.

Działają one jak niewidoczne kontenery, w których umieszczane są inne widoki i grupy widoków.

Tę kategorię reprezentują następujące klasy:

- Linear Layout,
- Relative Layout,
- Constraint Layout,
- Table Layout,
- Frame Layout,
- Web View,
- List View,
- Grid View.

Niektóre z nich mają swoje konkretne przeznaczenie.
Tak jak np. Web View, który jest kontenerem przeznaczonym do wyświetlania elementów renderowanych przez silnik przeglądarki internetowej wbudowany w Androida,

Layouty – ViewGroup

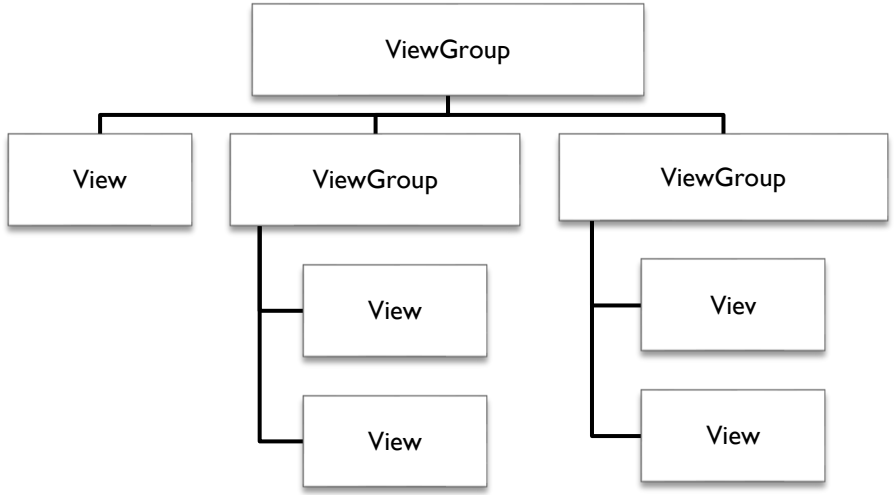
W tym momęcie należy jednak skupić się na elementach klasy ViewGroup określanych jako layouty.

Layout określa strukturę interfejsu użytkownika w aplikacji.

Sam zwykle jest transparentny, lecz definiuje sposób w jaki wszystkie elementy rozmieszczane są na ekranie.

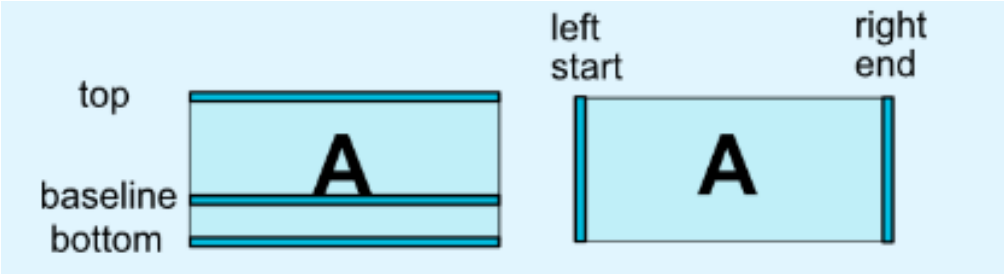
Wszystkie elementy w layoucie są zbudowane przy pomocy hierarchii widoków lub grup widoków.

Przykładowa hierarchia elementów typu View i ViewGroup



Constraint Layoutu

Constreint Layout daje możliwość podczepienia każdej z krawędzi widoku do jednej z dwóch krawędzi dowolnego innego widoku, linii pomocniczej lub krawędzi ekranu.



Źródło: https://developer.android.com

Podstawowe polecenia:

- layout_constraintTop_toBottomOf
- layout_constraintBottom_toBottomOf
- layout_constraintBottom_toTopOf
- layout_constraintTop_toTopOf
- layout_constraintStart_toStartOf
- layout_constraintStart_toEndOf
- layout_constraintEnd_toEndOf
- layout_constraintEnd_toStartOf

Parametrem każdego z tych poleceń jest Id innego widoku

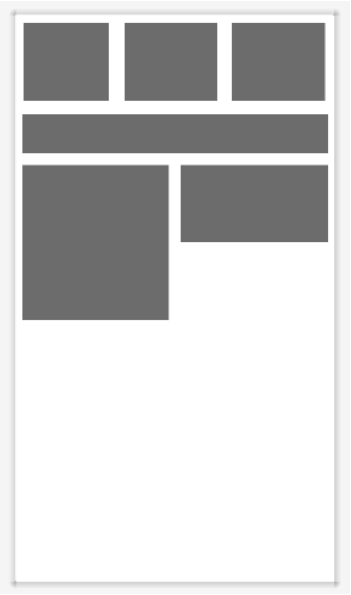
Linear Layoutu

Najprostszym układem (elementem typu View Group) jest Linear Layout.

Jest to widok, który rozkłada wszystkie elementy potomne pionowo lub poziomo.

Linear Layout może być zagnieżdżony, więc za pomocą kombinacji poziomych i pionowych layoutów tego typu można tworzyć różnego rodzaju, niekiedy nawet bardzo skomplikowane, układy tabelaryczne

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <ImageView…/>
        <ImageView…/>
        <ImageView…/>
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <ImageView…/>
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <ImageView…/>
        <ImageView…/>
    </LinearLayout>
</LinearLayout>
```



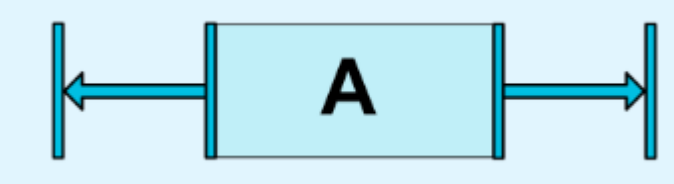
Przykład układu ekranu wykonanego za pomocą zagnieżdżonych układów Linear Layout

Constraint Layoutu

Wyśrodkowanie vs. rozciągnięcie na cały ekran

wrap_content – „dopasuj się do zawartości”

Wyśrodkowanie w poziomie




```
android:layout_width="wrap_content"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
```

Dopasowanie do szerokości ekranu

Źródło: https://developer.android.com

```
android:layout_width=„0dp”
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintLeft_toLeftOf="parent
```

parent – „rodzic” kontener, w którym znajduje się widok
(najczęściej oznacza cały ekran)

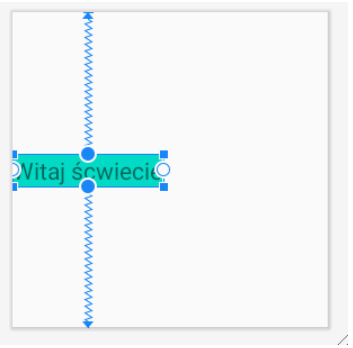
Ustawienie szerokości obiektu na 0dp oznacza, że powinna być ona wyliczona z innych ustawień

Constraint Layoutu

Wyśrodkowanie vs. rozciągnięcie na cały ekran

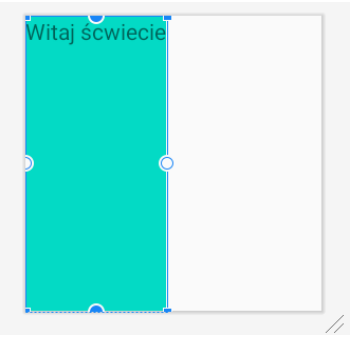
Wyśrodkowanie w pionie

```
android:layout_height="wrap_content"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintTop_toTopOf="parent"
```



Dopasowanie do wysokości ekranu

```
android:layout_height="0dp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintTop_toTopOf="parent"
```

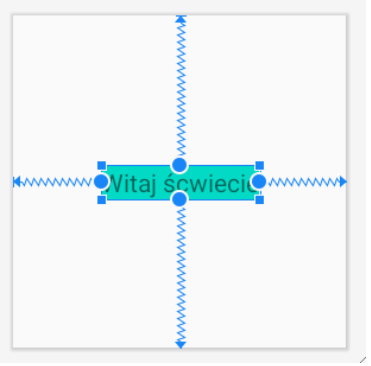


Ustawienie wysokości obiektu na 0dp oznacza, że powinna być ona wyliczona z innych ustawień

Constraint Layoutu

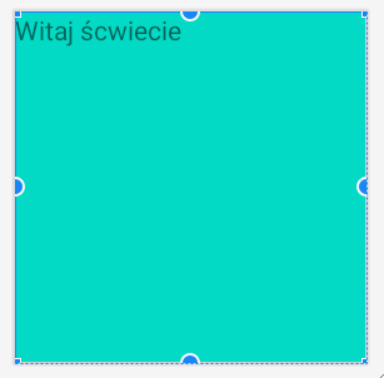
Wyśrodkowanie

```
android:layout_height="wrap_content"
android:layout_width="wrap_content"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
```



Rozciągnięcie na cały ekran

```
android:layout_height="0dp"
android:layout_width="0dp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
```



Ustawienie wysokości i szerokości obiektu na 0dp oznacza, że powinny być one wyliczone z innych ustawień

Constraint Layoutu

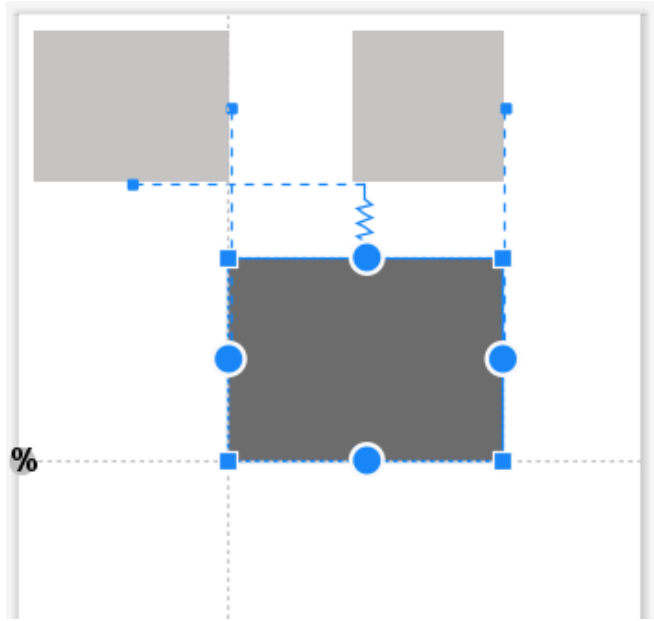
Guideline – linia pomocnicza - pozwala na ustawienie linii odniesienia do pozycjonowania pozostałych widoków. Może zostać ustawiona procentowo lub na konkretną wartość.

```
<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintGuide_begin="20dp" />
```

```
<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintGuide_percent="0.5" />
```

Constraint Layoutu

```
<ImageView
    android:id="@+id/el_03"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginTop="50dp"
    app:layout_constraintLeft_toRightOf="@id/el_01"
    app:layout_constraintRight_toRightOf="@id/el_02"
    app:layout_constraintTop_toBottomOf="@id/el_01"
    app:layout_constraintBottom_toTopOf="@id/gl"
/>
```

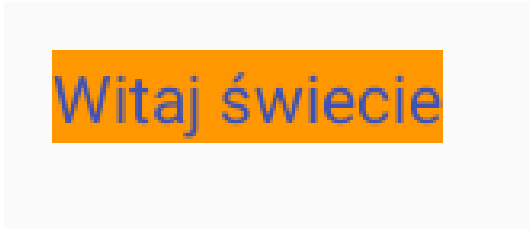


Przykład pozycjonowania widoku w układzie Constraint Layout

Definiowanie wyglądu elementów

Kolory

```
android:background="#FF9800"
android:textColor="#3F51B5"
```



Jednostki

px – piksele: odradzane ze względu na różną gęstość pikseli na wyświetlaczach.

mm – milimetry: jednostka bazująca na wielkości wyświetlacza.

in - cale: jednostka bazująca na wielkości wyświetlacza.

pt – punkty: 1pt = 1/72 cala

dp – Density-independent Pixels: polecana - jednostka niezależna od gęstości pikseli. Uniezależnia element layoutu od rozdzielczości oraz wielkości wyświetlacza. Jednostka ta bazuje na wyświetlaczu 160dpi (1dp jest odpowiednikiem 1 piksela przy gęstości 160dpi) Stosunek piksel-dp jest za każdym razem dobierany do gęstości wyświetlacza.

sp - Scale-independent Pixels: Jednostka podobna do dp. Podczas skalowania pod uwagę brane są również ustawienia użytkownika (Settings.System.FONT_SCALE). Należy stosować do definiowania wielkości czcionek oraz wszystkich wymiarów z nimi związanych.

Definiowanie wyglądu elementów

Model pudełkowy widoku (elementu typu view)

Od modelu pudełkowego znanego z HTML i CSS różni się on brakiem elementu „border” czyli możliwości definiowania obramowania. Wynika to stąd, że mamy do czynienia z predefiniowanymi kontrolkami.

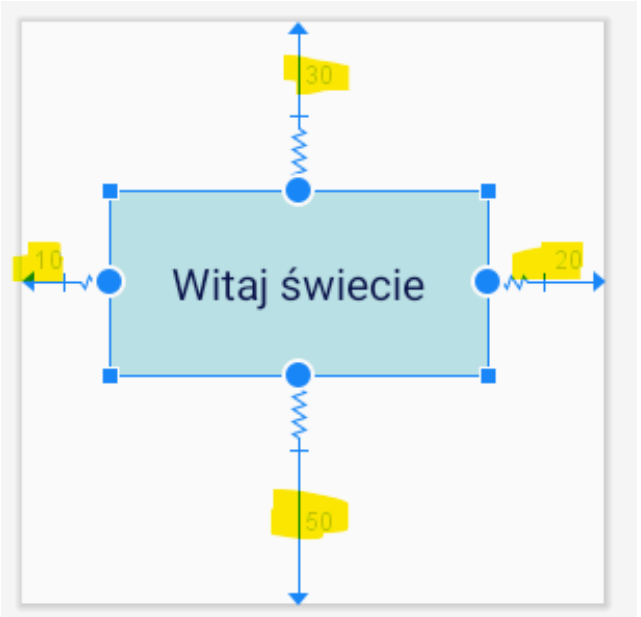
Definiowanie własnych obramowań jest oczywiście możliwe, lecz nieco bardziej skomplikowane (wymaga przedefiniowania wyglądu kontrolki)

Definiowanie wyglądu elementów

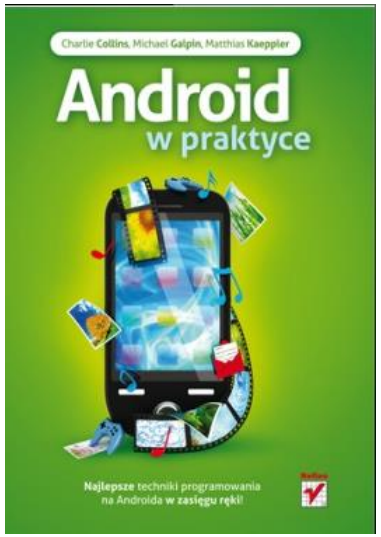
Model pudełkowy widoku (elementu typu view)

```
android:padding="20dp"
android:layout_margin="10dp"
```

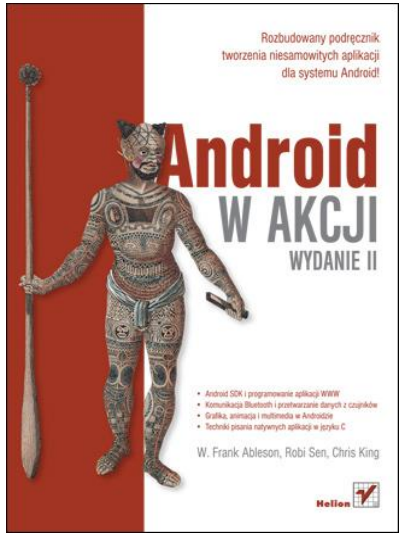
```
android:padding="20dp"
android:layout_marginLeft="10dp"
android:layout_marginRight="20dp"
android:layout_marginTop="30dp"
android:layout_marginBottom="50dp"
```

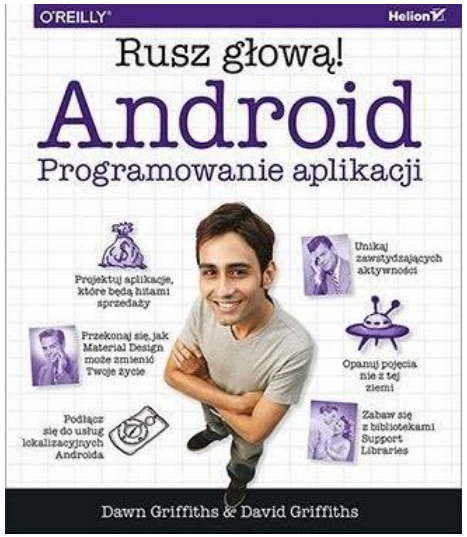


Literatura



https://developer.android.com





https://javastart.pl/baza-wiedzy/android/

https://forum.android.com.pl