

Programowanie Cross-Platformowe

Technologie, architektura i porównanie
z programowaniem natywnym

dr inż. Mateusz Pomianek

Zawartość prezentacji

1

Programowanie cross-platformowe

Definicje, rynek i statystyki adopcji

3

Architektura aplikacji mobilnych

Struktura i wzorce architektoniczne

5

Flutter i Dart FFI

Framework Google i interfejs do kodu natywnego

7

Python na urządzeniach mobilnych

Frameworki Kivy i BeeWare

9

Solar2D

Silnik gier 2D w

Lua

11

Podsumowanie i wnioski

Rekomendacje i trendy przyszłościowe

2

natywne vs cross-platformowe

Porównanie podejść, zalety i wady

4

React Native

Framework Meta do tworzenia natywnych aplikacji

6

Progressive Web Apps (PWA)

Aplikacje webowe z funkcjonalnością natywną

8

RubyMotion

Ruby dla aplikacji natywnych iOS i Android

10

Rust FFI i gomobile

Rust i Go w rozwoju mobilnym

01

Wprowadzenie do programowania cross-platformowego

Definicje, rynek i statystyki adopcji

Czym jest programowanie cross-platformowe?

Definicja

Programowanie cross-platformowe to podejście polegające na tworzeniu aplikacji, które mogą działać na wielu platformach (iOS, Android) z wykorzystaniem **jednej bazy kodu**. Frameworki cross-platformowe umożliwiają deweloperom pisanie kodu raz i wdrażanie go na różne systemy operacyjne.

Celem jest redukcja kosztów, skrócenie czasu wdrożenia oraz zapewnienie spójnego doświadczenia użytkownika na wszystkich platformach.

Kluczowe frameworki

React Native

Meta, JavaScript

Flutter

Google, Dart

Xamarin/.NET MAUI

Microsoft, C#

Ionic/Cordova

WebView-based

Statystyki rynkowe (2025)

Wartość rynku Frameworki cross-platformowe	\$50 mld
Wzrost CAGR 2025-2033	20%
Udział w rynku Wszystkich aplikacji mobilnych	42%
Użytkownicy smartfonów Na świecie w 2025	7 mld+

*"Cross-platform development can reduce costs by **25-50%** compared to native development"*

— Mobisoft Infotech, 2025

Ewolucja technologii cross-platformowych

1

Era WebView (2010-2014)

Cordova, PhoneGap: Aplikacje hybrydowe opakowujące strony web w natywny kontener. Problemy: niska wydajność, brak natywnego UX, ograniczony dostęp do sprzętu.

2

Podejście hybrydowe (2014-2018)

Ionic, Framework7: Ulepszone UI komponenty, lepsza integracja z natywnymi funkcjami. Nadal ograniczenia wydajnościowe związane z WebView.

3

Mostek natywny (2015-2020)

React Native: Komponenty JS komunikują się z natywnymi modułami przez mostek. Lepsza wydajność, natywny wygląd, ale opóźnienia związane z mostkiem.

4

Kompilacja AOT (2018-obecnie)

Flutter: Kompilacja Ahead-of-Time do kodu maszynowego, własny silnik renderujący. Wydajność bliska natywnej, płynne animacje 60/120 FPS.

5

Nowoczesna architektura (2020-obecnie)

React Native New Architecture, SwiftUI/React Server Components: Synchroniczne wywołania natywne, eliminacja mostka, wsparcie dla wielu platform (web, desktop).

Wniosek: Nowoczesne frameworki eliminują problemy wydajnościowe starszych rozwiązań, oferując wydajność zbliżoną do natywnej przy zachowaniu zalet cross-platform.

02

Programowanie natywne vs cross-platformowe

Porównanie podejść, zalety i wady

Definicja programowania natywnego

Czym jest programowanie natywne?

Programowanie natywne polega na tworzeniu aplikacji specyficznych dla danej platformy, wykorzystując **natywne języki i narzędzia** stworzone dla konkretnego systemu operacyjnego.

Zapewnia pełną integrację z ekosystemem platformy, optymalizację grafiki, efektywność energetyczną oraz natywne komponenty UI, które czują się naturalnie dla użytkowników.

Technologie natywne



iOS

Swift, Objective-C, Xcode



Android

Kotlin, Java, Android Studio

Statystyki (2025)

Udział rynku	~66%
Aplikacje natywne dominują	
Swift - popularność	4.91%
9. miejsce w rankingu	
Swift - top App Store	53%
Top non-game apps	
Objective-C	2.39%
Spadek popularności	

Kluczowe cechy

- ✓ Najlepsza wydajność i płynność
- ✓ Pełny dostęp do API i sprzętu
- ✓ Natywne UX/UI zgodne z wytycznymi
- ✓ Natychmiastowe wsparcie nowych funkcji OS

Porównanie: natywne vs cross-platform

Kryterium	Natywne	Cross-platform
Wydajność	Najlepsza, bo bezpośredni dostęp do sprzętu	~ Dobra - Flutter bliski natywnemu, RN z mostkiem
Czas rozwoju	4-6 miesięcy na platformę	2-4 miesiące dla wszystkich platform
Koszty	Wyższe - dwa zespoły, dwie bazy kodu	Niższe o 25-50% - jeden zespół, jedna baza kodu
Reużywalność kodu	0% - osobne bazy kodu	70-95% - wspólny kod
Dostęp do sprzętu	Pełny - natywne API	Ograniczony - przez wtyczki/natywne moduły
UX/UI	Natywny - zgodny z wytycznymi platformy	Spójny - może wymagać dostosowania
Utrzymanie	Trudniejsze - dwie bazy kodu	Łatwiejsze - jedna baza kodu
Aktualizacje OS	Natychmiastowe	Opóźnione - zależne od frameworka

Zastosowanie natywne: Gry, AR/VR, aplikacje medyczne, IoT, wymagające pełnej wydajności

Zastosowanie cross-platform: MVP, aplikacje treściowe, e-commerce, startupy, ograniczony budżet

Walety i zady programowania natywnego

+ Zalety

Najlepsza wydajność

Bezpośredni dostęp do procesora, GPU i pamięci. Brak narzutu frameworka.

Pełny dostęp do API

Natychmiastowy dostęp do wszystkich funkcji systemu i sprzętu.

Natywne UX/UI

Zgodność z wytycznymi platformy, intuicyjne gesty i animacje.

Lepsza obsługa offline

Zaawansowane możliwości przechowywania danych lokalnie.

Wbudowane bezpieczeństwo

Natywne mechanizmy szyfrowania, Keychain, Secure Enclave.

- Wady

Wyższe koszty

Konieczność utrzymywania dwóch zespołów i dwóch baz kodu.

Dłuższy czas rozwoju

4-6 miesięcy na platformę vs 2-4 miesiące cross-platform.

Złożoność zespołowa

Wymaga ekspertów od iOS i Android, trudniejsza komunikacja.

Ograniczony zasięg MVP

Trudniejsze testowanie produktu na wielu platformach jednocześnie.

Podwójne utrzymanie

Bug fixes i aktualizacje muszą być implementowane osobno.

Statystyka:

usuwa aplikacje o słabej wydajności - natywne podejście minimalizuje to ryzyko.

Walety i zady programowania cross-platformowego

+ Zalety

Jedna baza kodu

70-95% reużywalności kodu między iOS i Android.

Szybszy czas wdrożenia

2-4 miesiące vs 4-6 miesięcy na platformę w podejściu natywnym.

Niższe koszty

Redukcja kosztów o **25-50%** , jeden zespół deweloperski.

Szerszy zasięg

Jednoczesne wdrożenie na iOS i Android, większa baza użytkowników.

Łatwiejsze utrzymanie

Jedna baza kodu = łatwiejsze bug fixes i aktualizacje.

- Wady

Problemy wydajnościowe

Mostek JS w React Native, operacje intensywne obliczeniowo.

Ograniczony dostęp do funkcji

Dostęp do funkcji specyficznych przez wtyczki lub natywne moduły.

Niespójności UI

Różne wymagania UI między platformami, konieczność dostosowania.

Zależność od frameworka

Opóźnienia w dostępie do nowych funkcji OS, ryzyko porzucenia.

Większy rozmiar aplikacji

Framework dodaje narzut do rozmiaru aplikacji.

Dowód komercyjny: React Native i Flutter wygenerowały

przychodu w Q4 2024.

03

Architektura aplikacji mobilnych

Struktura i wzorce architektoniczne

Architektura aplikacji natywnych

Struktura warstwowa

Aplikacje natywne komunikują się **bezpośrednio** z systemem operacyjnym przez natywne API i SDK platformy. Architektura jest zoptymalizowana pod konkretną platformę.

1 UI Layer

Natywne komponenty UI (*UIKit/SwiftUI dla iOS, Views/Jetpack dla Android*)

2 Business Logic Layer

Logika aplikacji, przetwarzanie danych, walidacja, zarządzanie stanem

3 Data Layer

Dostęp do baz danych, API sieciowe, lokalne przechowywanie, cache

Wzorce architektoniczne

MVC (Model-View-Controller)

Tradycyjny wzorzec, szeroko stosowany w iOS

MVP (Model-View-Presenter)

Lepsza separacja, testowalność

MVVM (Model-View-ViewModel)

Reaktywne podejście, data binding

VIPER (iOS) / MVI (Android)

Zaawansowana separacja, jednokierunkowy przepływ danych

Kluczowe cechy

- ✓ Bezpośredni dostęp do sprzętu
- ✓ Optymalizacja pod platformę
- ✓ Natywne zarządzanie pamięcią
- ✓ Pełna kontrola nad wątkami

Architektura aplikacji cross-platformowych

React Native Architecture

JavaScript Thread

Wykonuje kod JS, zarządza logiką aplikacji

Bridge (Mostek)

Komunikacja asynchroniczna między JS a natywnymi modułami

Native Modules

Natywne komponenty UI i API platformy

Shadow Thread

Obliczenia layoutu (Yoga engine)

Nowa architektura: Fabric (renderer), TurboModules, JSI - eliminują mostek

Warstwa abstrakcji

Frameworki cross-platformowe tworzą **warstwę abstrakcji** między kodem aplikacji a systemem operacyjnym. Tłumaczą wywołania frameworka na natywne API platformy.

Flutter Architecture

Framework Layer (Dart)

Widgety, Material/Cupertino, gesty, animacje

Engine Layer (C++)

Skia (rendering), Dart VM, text rendering

Embedder Layer

Integracja z platformą, natywne API

Kompilacja AOT: Kod Dart kompilowany bezpośrednio do kodu maszynowego

Kluczowa różnica

React Native: Mostek JS-Native (asynchroniczny)

Flutter: Bezpośrednia kompilacja, własny silnik renderujący

04

React Native

Framework Meta do tworzenia natywnych aplikacji

Podstawy React Native

Czym jest React Native?

React Native to framework stworzony przez **Meta (Facebook)**, umożliwiający tworzenie natywnych aplikacji mobilnych przy użyciu **JavaScript/TypeScript** i React.

Zasada działania: komponenty JavaScript komunikują się z natywnymi modułami przez **mostek (bridge)**, który serializuje i deserializuje dane między światem JS, a natywnym.

Kluczowe cechy

Reużywalność kodu

70-90% wspólnego kodu

Hot Reload

Błyskawiczne odświeżanie

Natywne komponenty

Prawdziwe natywne UI

Ekosystem npm

2M+ pakietów

Popularne aplikacje

[Facebook](#)[Instagram](#)[Airbnb](#)[Discord](#)[Shopify](#)

Statystyki (2025)

Udział w rynku cross-platform	35%
Deweloperzy używający RN	32%
Wartość rynku RN	\$350M
GitHub stars	120k

Przykładowy kod

```
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      <Text>Hello React Native!</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
});

export default App;
```

Wydajność i architektura React Native

Nowa architektura React Native

Nowa architektura eliminuje wąskie gardła mostka przez wprowadzenie **JSI (JavaScript Interface)** - synchronicznego mostu między JS, a natywnym kodem.

Fabric

Nowy renderer UI, synchroniczny dostęp do natywnych komponentów

TurboModules

Leniwe ładowanie natywnych modułów, lepsza wydajność

JSI

Bezpośredni dostęp do natywnych obiektów z JS

Wydajność (benchmarks)

Czas uruchomienia iOS vs Native 0.9s, Flutter 1.2s	1.8s
Czas uruchomienia Android vs Native 1.1s, Flutter 1.4s	2.1s
Zużycie pamięci iOS vs Native 38MB, Flutter 45MB	68MB
FPS rendering (60 FPS) vs Native 99%, Flutter 98%	89%

Zalety

- Ogromny ekosystem npm (*2M+ pakietów*)
- Łatwość znalezienia deweloperów JS
- Współdzielenie kodu z React Web
- Szybkie prototypowanie z Expo

Wady

- Mostek może powodować opóźnienia
- Zależność od natywnych modułów
- Problemy z złożonymi animacjami

05

Flutter i Dart FFI

Framework Google i interfejs do kodu natywnego

Flutter: podstawy

Czym jest Flutter?

Flutter to framework stworzony przez **Google**, używający języka **Dart**. Kompiluje kod do **kodu maszynowego (AOT)**, eliminując mostek i oferując wydajność bliską natywnej.

Flutter używa **własnego silnika renderującego Skia**, co zapewnia spójny wygląd i płynność na wszystkich platformach. Filozofia: "Everything is a widget".

Kluczowe cechy

Kompilacja AOT

Do kodu maszynowego

Widget-based UI

Wszystko jest widgetem

Hot Reload

0.4s - najszybszy

Multi-platform

Mobile + Web + Desktop

Popularne aplikacje

Google Pay

Alibaba

BMW

eBay

Reflectly

Statystyki (2025)

Udział w rynku cross-platform
Lider rynku

46%

Deweloperzy Flutter
Wzrost 10% miesięcznie

2-5M

GitHub stars
vs React Native 120k

170k

Wzrost na Google Play
w 2025 roku

+30%

Przykładowy kod

```
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Text('Hello Flutter!'),
        ),
      ),
    );
  }
}
```

Flutter: wydajność i porównanie

Benchmarks wydajności

Czas uruchomienia iOS **1.2s**

vs Native 0.9s, React Native 1.8s

Czas uruchomienia Android **1.4s**

vs Native 1.1s, React Native 2.1s

Zużycie pamięci iOS **45MB**

vs Native 38MB, React Native 68MB

FPS rendering (60 FPS) **98%**

vs Native 99%, React Native 89%

Flutter wygrywa

- ✓ Wydajność (20-30% szybszy)
- ✓ Spójność UI
- ✓ Szybkość rozwoju
- ✓ Multi-platform

React Native wygrywa

- ✓ Dostępność deweloperów
- ✓ Ekosystem npm
- ✓ Krzywa uczenia
- ✓ Web + mobile sharing

Rekomendacja 2025

Dla większości projektów: **Flutter** . Dla zespołów z doświadczeniem JS: **React Native** .

Porównanie kosztów i czasu

Koszt MVP **\$65K**

vs React Native \$73K

Czas rozwoju **304h**

vs React Native 343h (13% szybciej)

Zużycie CPU (idle) **2%**

vs React Native 4%

Zużycie baterii (1h) **8%**

vs React Native 11%

Dart FFI: Foreign Function Interface

Czym jest Dart FFI?

Dart FFI umożliwia wywoływanie funkcji C/C++ bezpośrednio z kodu Dart z wydajnością bliską natywnej. Eliminuje narzut serializacji związany z MethodChannel.

Mechanizm: `DynamicLibrary.open`, `lookup`, `asFunction`. Overhead FFI to zaledwie **~100ns na wywołanie** - znacznie mniej niż MethodChannel z JSON.

Zastosowania FFI

Obliczenia numeryczne

Algebra liniowa, symulacje fizyczne

Przetwarzanie obrazu/audio

Kodeki, filtry, efekty

Kryptografia

Szyfrowanie, hashowanie, podpisy

Machine Learning

Integracja z bibliotekami C/C++

Najlepsze praktyki

- Minimalizuj przejścia FFI - wsadowe operacje
- Używaj POD structs do przekazywania danych
- Unikaj częstego tworzenia `NativeCallback`
- Reużywaj buforów, unikaj `malloc/free` w pętlach
- Używaj izolatów dla długich operacji

Przykładowy kod FFI

```
import 'dart:ffi';
// Zdefiniuj typy funkcji
typedef CAdd = Int32 Function(Int32 a, Int32 b);
typedef DartAdd = int Function(int a, int b);
void main() {
  // Otwórz bibliotekę
  final dylib = DynamicLibrary.open('libmath.so');
  // Pobierz funkcję
  final DartAdd add = dylib
    .lookup<NativeFunction<CAdd>>('add')
    .asFunction();
  // Wywołaj funkcję
  final int result = add(2, 3);
  print(result); // 5
}
```

Porównanie z MethodChannel

MethodChannel

JSON serialization, opóźnienia,
wyższy narzut

FFI

~100ns/call, bezpośredni dostęp,
najwyższa wydajność

06

Progressive Web Apps

Aplikacje webowe z funkcjonalnością natywną

PWA: definicja i komponenty

Czym jest PWA?

Progressive Web App (PWA) to aplikacja webowa wykorzystująca nowoczesne technologie webowe do oferowania **doświadczenia podobnego do aplikacji natywnych**, bez konieczności instalacji ze sklepu z aplikacjami.

PWA łączą najlepsze cechy stron webowych (dostępność, indeksowalność) z funkcjonalnościami aplikacji mobilnych (offline, push notifications, dostęp do sprzętu).

Kluczowe komponenty

1 Service Workers

Skrypty działające w tle, obsługujące cache, offline, push notifications

2 Web App Manifest

JSON opisujący aplikację: nazwa, ikony, kolory, tryb wyświetlania

3 HTTPS

Wymagane dla Service Workers, zapewnia bezpieczeństwo

Statystyki adopcji (2025)

Strony używające funkcji PWA	24.5%
Pełne kryteria instalacji	3.3-3.5%
Duże marki używające PWA	60%
Przychód PWA 2024-25	\$10.7B
Prognoza 2027	\$23B

Popularne PWA

Starbucks

Pinterest

Uber

Twitter Lite

Flipkart

PWA: wydajność i ograniczenia

Wydajność PWA

4x szybsze ładowanie

W porównaniu do tradycyjnych stron mobilnych

-50% bounce rate

Redukcja współczynnika odrzuceń

+36% konwersja

Średni wzrost w e-commerce

Case studies

Starbucks

148MB → 600KB, działa na 2G

Pinterest

+60% engagement, +44% ad revenue

Flipkart Lite

+40% re-engagement rates

Wniosek: PWA oferują najniższy próg wejścia i doskonałą wydajność dla treściowych aplikacji, ale mają ograniczenia na iOS i w dostępie do zaawansowanych funkcji sprzętowych.

Ograniczenia na iOS

Tylko Safari

~80% użytkowników iOS używa Safari

Brak automatycznego promptu

Użytkownik musi dodać ręcznie przez menu Share

Limit storage: 50MB

Ograniczone przechowywanie offline

Brak background sync

Brak synchronizacji w tle

Ograniczony dostęp do sprzętu

Brak Bluetooth, ARKit, Face ID, Touch ID

Scorecard przeglądarek

Chrome 131	97/100
Safari 26	86/100
Firefox 138	82/100

07

Python na urządzeniach mobilnych

Frameworki Kivy i BeeWare

Kivy i BeeWare: frameworki Python



Kivy

Kivy to open-source framework do tworzenia aplikacji z **niestandardowym UI**. Używa OpenGL ES 2 do akceleracji GPU.

Zalety

Szybka grafika GPU, custom UI, duża społeczność

Wady

Trudniejsza krzywa uczenia, niestandardowy UI

Inne narzędzia Python

Chaquopy

Android development z Python

Pyjnius

Wywoływanie Java API z Python

Buildozer

Pakowanie aplikacji Python na Android/iOS

Zastosowanie: Python mobilny to nisza - najlepiej sprawdza się w aplikacjach wymagających integracji z data science, ML lub szybkiego prototypowania.



BeeWare

BeeWare to open-source framework z naciskiem na **natywne UI**. Bardziej python-owy, łatwiejszy w nauce.

Zalety

Natywne UI, pełne wsparcie platform, łatwiejszy w nauce

Wady

Mniejsza społeczność, ograniczona customizacja

Porównanie Kivy vs BeeWare

Cecha	Kivy	BeeWare
UI	Niestandardowy	Natwywny
Grafika	OpenGL ES 2	Platforma
Krzywa uczenia	Trudniejsza	Łatwiejsza
Spółeczność	Duża	Rosnąca

Cocos2d-x: wydajne gry 2D w C++

Cocos2d-x

Cocos2d-x to framework do tworzenia gier 2D w języku C++ z możliwością kompilacji krzyżowej na iOS, Android, Web, Windows, macOS i Linux. Powstał jako fork Cocos2d (*Python/Objective-C*).

Kluczowe cechy

- Silnik renderujący OpenGL ES / Metal / Vulkan
- Narzędzie Cocos Creator – wizualny edytor scen
- Wsparcie dla JavaScript/Lua poprzez wiązania (bindings)
- Bardzo mały rozmiar wynikowego APK (~3–5 MB)

Największe tytuły

Clash of Clans

Badland

wczesne wersje Hearthstone

Zalety

- Najwyższa wydajność (C++ + natywny rendering)
- Niskie zużycie baterii i pamięci
- Świetny do gier 2D z dużą liczbą obiektów

Wady

- Krzywa uczenia (C++, manualne zarządzanie pamięcią)
- Mniej popularny na Zachodzie

Statystyki

Liczba gier wyprodukowanych: **ponad 1,6 mln**

Udział w chińskim rynku gier mobilnych: **~25%**

Przykładowy kod

```
#include "cocos2d.h",  
USING_NS_CC;  
  
class MainScene : public Scene  
{public:  
    bool init() override {  
        auto label = Label::createWithTTF("Hello Cocos",  
        "arial.ttf", 24);  
        label->setPosition(100, 100);  
        addChild(label);  
        return true;  
    }  
    CREATE_FUNC(MainScene);};
```

09

Unity, Gadot, Solar2D, Defold, Phaser

Silnik gier 2D w Lua

Unity: najpopularniejszy silnik gier cross-platform

Czym jest Unity?

Unity to najszerzej stosowany profesjonalny silnik gier na świecie. Umożliwia tworzenie gier 2D, 3D, AR/VR na ponad 25 platform, w tym iOS, Android, Web, konsole, PC i urządzenia wearable.

Kluczowe cechy

- Języki: C# (główny), C++ (pluginy natywne), Burst (kompilator wysokiej wydajności)
- Renderowanie: Universal Render Pipeline (URP), High-Definition RP (HDRP)
- Mechanizmy: ECS (Entity Component System), DOTS (Data-Oriented Tech Stack)
- Asset Store z ponad 80 tysiącami zasobów

Przykładowe tytuły

Genshin Impact

Among Us

Hollow Knight

Pokémon GO

Zalety

- Ogromny ekosystem i wsparcie
- Wspaniała dokumentacja i tutoriale
- Wydajność bliska natywnej (z DOTS)
- Jedna baza kodu dla wszystkich platform

Wady

- Rosnące koszty licencji (Unity Pro od ~\$2200/rok)
- Częste zmiany polityki cenowej (kontrowersje 2023/2024)
- Duży rozmiar aplikacji (minimalny APK ~20 MB)

Statystyki

Udział w rynku gier mobilnych: **~55%**
Liczba deweloperów Unity: **~2,8 mln/mc**
Gry zbudowane w Unity: **ponad 70% top 1000 gier mobile**

Platforma	Unity (URP)	Natywne (Swift/Kotlin)	Różnica
Czas startu (iOS)	1,4 s	0,9 s	+55%
Zużycie RAM	120 MB	78 MB	+54%
FPS (60 klatek)	59,2 FPS	60 FPS	praktycznie identyczny

Godot: silnik gier 2D i 3D open-source

Czym jest Godot?

Godot to w pełni open-source'owy silnik gier obsługujący 2D i 3D, rozwijany przez Fundację Godot. Umożliwia tworzenie gier na platformy: Windows, macOS, Linux, Android, iOS, Web (HTML5), a także konsole (przez partnerów).

Kluczowe cechy

- Własny język skryptowy **GScript** (podobny do Pythona)
- Możliwość używania **C#**, **C++** lub **VisualScript**
- Architektura scen i węzłów (SceneTree)
- Wbudowane narzędzia: edytor animacji, tilemap, shadery, debugger

Największe tytuły

Cassette Beasts

Cruelty Squad

Sonic Colors: Ultimate *(współpraca)*

Zalety

- Całkowicie darmowy i bez opłat licencyjnych
- Lekki (instalacja < 100 MB)
- Świetna dokumentacja i społeczność

Wady

- Mniej zaawansowany rendering 3D niż Unity/Unreal
- Mniejszy ekosystem assetów

Statystyki

GitHub stars	~75k
Aktywni użytkownicy miesięcznie	~500k
Udział w rynku silników gier indie	~18%

Przykładowy kod

```
Extends Node2D
func _ready():
    print("Witaj w Godot!")
    var scena = preload("res://scs/lvl.tscn").instantiate()
    add_child(nowa_scena)

func _process(delta):
    position += Vector2(100, 0) * delta
```

Solar2D: silnik gier Lua

Czym jest Solar2D?

Solar2D (dawniej Corona SDK) to framework do tworzenia **gier 2D i aplikacji w Lua**.
Darmowy, open-source, z ponad 1000 API.

Solar2D oferuje **real-time testing** - kod aktualizuje się natychmiast na urządzeniu przez sieć lokalną, bez konieczności rekompilacji.

Wspierane platformy

iOS	Android	Kindle
Windows	macOS	Linux
Apple TV	Fire TV	Android TV

Popularne gry

Flappy Bird

Match 3 Space RPG

Corona Cannon

Funkcje

- Ponad 1000 API
- Pluginy (monetyzacja, analytics)
- Solar2D Playground (online)
- Solar2D Native (natywne rozszerzenia)
- CoronaCards (embed w natywnych)
- Fizyka, dźwięk, grafika 2D
- Sieć i multiplayer

Przykładowy kod

```
local widget = require("widget")  
  
-- Obsługa zdarzenia przycisku  
local function handleButtonEvent(event)  
    if event.phase == "ended" then  
        print("Button clicked")  
    end  
end  
  
-- Stwórz przycisk  
local button = widget.newButton({  
    label = "Click Me",  
    onEvent = handleButtonEvent  
})  
  
-- Dodaj do sceny  
display.getCurrentStage():insert(button)
```

Zalety

- ✓ Darmowy i open-source
- ✓ Szybki development
- ✓ Lua - łatwa do nauki
- ✓ Real-time testing

Defold: silnik gier Lua

Czym jest Defold?

Defold to darmowy, wydajny silnik gier 2D (od twórców *Candy Crush*) stworzony przez King (*Activision Blizzard*). Obecnie rozwijany jako open-source. Ukierunkowany na gry mobilne i webowe.

Kluczowe cechy

- Język skryptowy: Lua
- Wydajny rendering 2D z potokiem opartym na komponentach
- Wbudowany edytor, debugger, profilowanie
- Bardzo mały runtime (~2 MB)

Używany przez

King

Rovio

Ragdoll Games

Statystyki

Liczba gier wyprodukowanych **ponad 40 000**
Miesięczna liczba aktywnych deweloperów **~20k**
Średni czas uruchomienia gry **<0,5s**

Przykładowy kod

```
#include "cocos2d.h",  
USING_NS_CC;  
  
class MainScene : public Scene  
{public:  
    bool init() override {  
        auto label = Label::createWithTTF("Hello Cocos",  
        "arial.ttf", 24);  
        label->setPosition(100, 100);  
        addChild(label);  
        return true;  
    }  
    CREATE_FUNC(MainScene);};
```

Zalety

- Bardzo niskie zużycie zasobów (idealne na słabsze urządzenia)
- Szybki hot-reload na urządzeniach mobilnych
- W pełni darmowy (brak tantiem)

Wady

- Mniejsza społeczność niż Unity/Godot
- Głównie 2D (3D *bardzo ograniczone*)

Phaser: Framework gier 2D w przeglądarce

Czym jest Phaser?

Phaser to darmowy framework do tworzenia gier 2D w JavaScript/TypeScript, działających w przeglądarce. Dzięki temu gry są automatycznie cross-platformowe (web, mobile, desktop) bez instalacji.

Kluczowe cechy

- Silnik renderujący: Canvas / WebGL
- System scen, fizyka (Arcade, Matter.js), animacje, dźwięk
- Obsługa WebAssembly dla wydajnych operacji
- Integracja z PWA (gry offline)

Używany przez

Disney

NASA

Miniclip

Statystyki

GitHub stars	~36k
NPM downloads	~2,5 mln miesięcznie
Używany przez	Disney, NASA, Miniclip
Udział w rynku gier HTML5	~40%

Przykładowy kod

```
#include "cocos2d.h",
USING_NS_CC;

class MainScene : public Scene
{
public:
    bool init() override {
        auto label = Label::createWithTTF("Hello Cocos",
            "arial.ttf", 24);
        label->setPosition(100, 100);
        addChild(label);
        return true;
    }
    CREATE_FUNC(MainScene);};
```

Zalety

- Brak instalacji, działa w każdej przeglądarce
- Bardzo szybkie prototypowanie
- Ogromna liczba przykładów i pluginów

Wady

- Wydajność gier złożonych (wielu obiektów) niższa niż natywnych
- Ograniczony dostęp do sprzętu (brak pełnego Bluetooth, NFC itp.)

10

Rust FFI, gomobile i RubyMotion

Rust i Go w rozwoju mobilnym

Rust FFI dla mobilnych aplikacji

Rust w rozwoju mobilnym

Rust FFI umożliwia integrację Rust z aplikacjami mobilnymi przez **bindings**. Rust oferuje wydajność C/C++ z gwarancjami bezpieczeństwa pamięci.

Wydajność

Zero-cost abstractions, wydajność C/C++

Bezpieczeństwo

Ownership model, brak null pointer dereferences

Współbieżność

Bez data races, bezpieczne wątki

Integracja platform

Android

JNI (Java Native Interface) + Rust FFI

iOS

swift-bridge - automatyczne generowanie bindings

Narzędzia

- rust-android-gradle - Gradle plugin
- cargo-apk - Budowanie APK
- rust-ios - Wsparcie iOS
- swift-bridge - Swift bindings
- cbindgen - Generowanie C headers

Zastosowania

Gry i grafika

Silniki gier, renderowanie

Kryptografia

Bezpieczne operacje kryptograficzne

Niskopoziomowy dostęp

System calls, hardware access

Wzorzec "twinning"

Każda klasa JVM/Swift ma odpowiednik w Rust połączony przez FFI

Gomobile: Go na urządzenia mobilne

Czym jest gomobile?

gomobile to narzędzie Go do kompilacji aplikacji na **Android** i **iOS**. Umożliwia pisanie aplikacji całkowicie w Go lub tworzenie bibliotek do użycia w natywnych aplikacjach. Dwie strategie: **all-Go native apps** lub **SDK applications** z bindings.

Instalacja i użycie

Instalacja

```
$ go install golang.org/x/mobile/cmd/gomobile@latest
$ gomobile init # Budowanie APK
$ gomobile build -target=android \
golang.org/x/mobile/example/basic # Generowanie bindings
$ gomobile bind -target=android ./package
```

Dostępne pakiety

App control

Asset management

Audio

OpenGL ES 2/3

Event management

Sprite

Wymagania

Go wersja

1.16+

macOS (iOS)

Xcode Command Line Tools

Android

Android SDK, NDK

Ograniczenia

- Brak natywnego UI
- Ograniczone wsparcie bibliotek
- Wymaga hybrydy z Kotlin/Swift
- Większy rozmiar binarki

Alternatywa: Fyne

UI toolkit w Go dla cross-platform GUI (Windows, macOS, Linux, Android, iOS)

RubyMotion, framework Ruby

Czym jest RubyMotion?

RubyMotion to toolchain do tworzenia **natywnych aplikacji w Ruby** dla iOS, Android i OS X. Został stworzony przez Laurent Sansonetti (były pracownik Apple).

RubyMotion implementuje **zunifikowany runtime**: obiekty Ruby zachowują się jak Objective-C (iOS) lub Java (Android), eliminując potrzebę mostka.

Kluczowe cechy

Kompilacja AOT

LLVM do kodu maszynowego

Zunifikowany runtime

Ruby ↔ Objective-C/Java

Terminal workflow

Bez IDE, ulubiony edytor

REPL + Debugger

Interaktywny development

Wsparcie platform

iOS 5.0-9.1

Android 3.2-6.0

ARM 32/64-bit

OS X

Specyfikacja

Rozmiar aplikacji

<1MB

Typowa aplikacja

Zarządzanie pamięcią

ARC-like

Automatyczne (iOS/OS X)

Garbage Collector

Java GC

Dla Android

CocoaPods

✓

Natywne wsparcie (iOS)

Przykładowy kod

```
class AppDelegate
  def application(application, didFinishLaunchingWithOptions:
options)
    @window =
UIWindow.alloc.initWithFrame(UIScreen.mainScreen.bounds)
    @window.rootViewController = UIViewController.alloc.init
    @window.makeKeyAndVisible
    true
  end
end
```

RubyMotion Starter: Darmowa wersja ze splash screenem

11

Podsumowanie i wnioski

Rekomendacje i trendy przyszłościowe

Porównanie wszystkich technologii

Technologia	Język	Wydajność	Koszty	Reużywalność	Zastosowanie
Natywne	Swift/Kotlin	★★★★★	Wysokie	0%	Gry, AR/VR, medyczne
Flutter	Dart	★★★★☆	Średnie	90-95%	Większość aplikacji
React Native	JavaScript	★★★☆☆	Średnie	70-90%	Zespoły JS, web+mobile
PWA	JS/HTML/CSS	★★☆☆☆	Niskie	100% (web)	MVP, treściowe
Python (Kivy/BeeWare)	Python	★★☆☆☆	Niskie	80-90%	Data science, prototypy
RubyMotion	Ruby	★★★★☆	Średnie	70-80%	Szybkie prototypy
Solar2D	Lua	★★★★☆	Niskie	95%+	Gry 2D
Rust FFI	Rust	★★★★★	Wysokie	60-70%	Wydajność krytyczna
gomobile	Go	★★★★☆	Średnie	70-80%	Logika biznesowa

Podsumowanie: React Native i Flutter dominują rynek cross-platform. PWA oferuje najniższy próg wejścia. Python/Ruby/Go/Rust to nisze z określonymi przypadkami użycia.

Kiedy wybrać którą technologię?

Natywne

Gdy potrzebujesz:

- Najwyższej wydajności (gry, AR/VR)
- Pełnego dostępu do sprzętu
- Aplikacji medycznych/IoT
- Zaawansowanych animacji 3D



Flutter

Gdy potrzebujesz:

- Najlepszego stosunku jakości do ceny
- Custom UI i animacji
- Multi-platform (mobile+web+desktop)
- Szybkiego developmentu



React Native

Gdy potrzebujesz:

- Zespołu z doświadczeniem JS
- Współdzielenia kodu z web
- Szybkiego prototypowania (Expo)
- Dostępu do ogromnego ekosystemu npm



PWA

Gdy potrzebujesz:

- Szybkiego MVP
- Aplikacji treściowych
- Ograniczonego budżetu
- Bez instalacji ze sklepu



Nisze specjalistyczne

Python: Data science, ML, prototypy

RubyMotion: Szybki prototyp, zespół Ruby

Solar2D: Gry 2D

Rust FFI: Wydajność krytyczna, kryptografia

gomobile: Logika biznesowa w Go

Trendy 2025-2026

- AI-powered PWAs
- WebAssembly (WASM) w PWA
- Flutter dla desktop i web
- React Native New Architecture
- Rosnąca adopcja Rust w mobile

**Dziękuję
za uwagę**

Koniec