WYKŁAD 8

Programowanie aplikacji mobilnych współpracujących z IoT

Programowanie aplikacji mobilnych
dr inż. Mateusz Pomianek

Internet of Things, Technologie i architektura

- Protokoły komunikacji
- Android & iOS + IoT
- Cloud Platforms
- Bezpieczeństwo
- Case Studies

Plan wykładu

Slajd 3
Wprowadzenie do IoT i rola aplikacji mobilnych

Slajd 6
Protokoły komunikacji (MQTT, CoAP, REST, WebSocket)

Slajd 10
Łączność bezprzewodowa (BLE, Wi-Fi, Zigbee)

Slajd 14
Programowanie Android & iOS dla IoT

Slajd 18
Platformy chmurowe i przetwarzanie danych

Slajd 22
Bezpieczeństwo i zarządzanie urządzeniami

Slajd 26
Testowanie i optymalizacja

Slajd 29
Case Studies i podsumowanie

Definicja i ekosystem Internetu Rzeczy (IoT)

"Sieć urządzeń fizycznych wyposażonych w czujniki, oprogramowanie i łączność umożliwiającą wymianę danych bez udziału człowieka"

75 mld
Urządzeń IoT do 2030 r.

4.6 TB/dzień
Danych generowanych

11 tys. USD
Wydatki na IoT na firmę/rok

18.5%
CAGR rynku IoT 2024–2030

Warstwy ekosystemu IoT

Urządzenia
Czujniki, aktuatory, mikrokontrolery

→

Łączność
Protokoły, sieci bezprzewodowe

→

Platforma
Cloud, analityka, storage

→

Aplikacje
Mobilne, webowe, dashboardy

Rola aplikacji mobilnych w systemach IoT

Interfejs użytkownika

Wizualizacja danych z czujników, sterowanie urządzeniami w czasie rzeczywistym, powiadomienia push.

Brama komunikacyjna

Telefon jako lokalny hub IoT, agregacja danych z urządzeń BLE/Zigbee przed wysłaniem do chmury.

Przetwarzanie lokalne

Edge computing na urządzeniu mobilnym, preprocessing danych przed transmisją, redukcja latencji.

Zarządzanie urządzeniami

Provisioning, konfiguracja OTA, monitoring stanu urządzeń, zarządzanie aktualizacjami firmware.

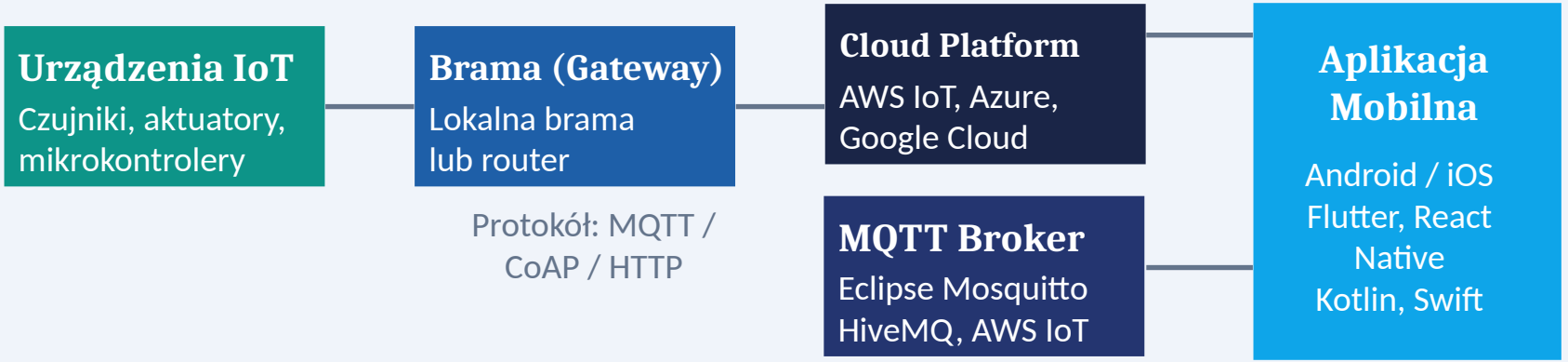
Analityka i raporty

Dashboardy, trendy historyczne, alerty progowe, eksport danych do systemów BI.

Bezpieczeństwo

Autentykacja użytkownika, autoryzacja dostępu do urządzeń, szyfrowanie komunikacji end-to-end.

Architektura Systemu IoT z aplikacją mobilną



Przepływ danych

1. Czujnik zbiera dane
2. Dane przesyłane przez MQTT/CoAP
3. Broker przekazuje wiadomości
4. Cloud przetwarza i przechowuje
5. Aplikacja mobilna prezentuje dane

Protokoły komunikacji w IoT

Wybór protokołu determinuje wydajność, niezawodność i skalowalność całego systemu IoT

Protokół	Transport	Model	QoS	Zastosowanie IoT
MQTT	TCP	Pub/Sub	0, 1, 2	Telemetria, smart home, przemysł
CoAP	UDP	Request/Response	0, 1	Urządzenia z ograniczonymi zasobami
HTTP/REST	TCP	Request/Response	brak	API, integracje, webowe dashboardy
WebSocket	TCP	Full-duplex	brak	Real-time UI, streaming danych
AMQP	TCP	Pub/Sub + Queuing	0, 1, 2	Systemy korporacyjne, kolejki
DDS	UDP/TCP	Pub/Sub P2P	0–5	Robotyka, systemy krytyczne

★ Rekomendowany dla nowych projektów IoT: MQTT (lekki, asynchroniczny, obsługa słabego łącza)

MQTT: Message Queuing Telemetry Transport

Broker

Centralny serwer pośredniczący (Mosquitto, HiveMQ, AWS IoT Core). Zarządza topicami i subskrypcjami.

Publisher

Urządzenie IoT publikujące dane na dany temat (topic). Np. sensor/temperatura/salon.

Subscriber

Klient (app mobilna) subskrybujący topic. Odbiera wiadomości w trybie push.

QoS Levels

0 – at most once (fire & forget)
1 – at least once (potwierdzenie)
2 – exactly once (pełna gwarancja)

Przykład: Android (Kotlin + MQTT)

```
val client = MqttAndroidClient(
    context, "tcp://broker.hivemq.com:1883",
    MqttClient.generateClientId()
)
val opts = MqttConnectOptions()
opts.isCleanSession = true
client.connect(opts, null, object :
IMqttActionListener {
    override fun onSuccess(token: IMqttToken?) {
        client.subscribe("sensor/#", 1)
    }
})
client.setCallback(object : MqttCallback {
    override fun messageArrived(
        topic: String, msg: MqttMessage) {
        val payload = msg.toString()
        // aktualizuj UI
    }
})
```

Hierarchia tematów (Topic Tree)

```
Home/ — korytarz/ — temp
              └— ruch
       └— salon/ — temp
                 └— wilgotnosc
       └— kuchnia/ — temp
```

CoAP: Constrained Application Protocol

RFC 7252, zaprojektowany specjalnie dla urządzeń embedded z ograniczonymi zasobami (RAM, CPU, energia)

MQTT vs CoAP

Cecha	MQTT	CoAP
Transport	TCP (niezawodny)	UDP (lekki, zawodny)
Model	Pub/Sub	Request/Response (REST-like)
Rozmiar nagłówka	2 bajty	4 bajty
Multicast	Nie	Tak (UDP multicast)
Observe	Subskrypcja na topic	CoAP Observe option (RFC 7641)
Energia	Wyższe zużycie	Niższe (UDP, uśpienie)

Kiedy używać CoAP? Urządzenia z baterią → mała pamięć RAM (< 10 KB) → sieci z dużą utratą pakietów → wymagana multicast discovery

Urządzenia z baterią · RAM < 10 KB · Sieci z dużą utratą pakietów · Wymagana multicast discovery

REST API dla IoT: HTTP i RESTful Services

Projektowanie REST API dla systemów IoT wymaga szczególnej uwagi na skalowalność i efektywność danych

Typowe endpointy REST API dla IoT

```
GET  /api/v1/devices
GET  /api/v1/devices/{id}
POST /api/v1/devices/{id}/command
GET  /api/v1/sensors/data
     ?from=2024-01-01&to=2024-12-
31
     &deviceId=dev001
     &metric=temperature
GET
/api/v1/devices/{id}/telemetry
     ?limit=100&offset=0
POST /api/v1/alerts/rules
PUT  /api/v1/devices/{id}/config
DEL  /api/v1/devices/{id}
```

Dobre praktyki REST dla IoT

Paginacja
limit/offset lub cursor-based. Niezbędne przy dużych zbiorach danych telemetrycznych

Kompresja
GZIP/Brotli, zmniejszenie danych do 80%, kluczowe dla urządzeń mobilnych

Wersjonowanie
URI versioning (/v1/, /v2/), kompatybilność wsteczna podczas aktualizacji

Rate Limiting
Throttling żądań (np. 1000 req/min), ochrona serwera przed przeciążeniem

HATEOAS
Linki nawigacji w odpowiedzi, samoopisowe API zmniejszające coupling

REST vs MQTT: HTTP jest lepszy dla operacji CRUD i integracji z systemami zewnętrznymi, natomiast MQTT przewyższa HTTP przy ciągłej telemetrii i urządzeniach z ograniczonymi zasobami.

WebSocket: komunikacja dwukierunkowa w czasie rzeczywistym

WebSocket (RFC 6455) umożliwia pełnodupleksową komunikację przez pojedyncze połączenie TCP, idealne dla real-time dashboardów IoT.

Nawiązanie połączenia WebSocket

1. HTTP Upgrade Request

```
GET /ws HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZQ==
```

2. Server 101 Switching

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Sec-WebSocket-Accept: s3pPL...
```

3. Full-duplex channel

Obie strony mogą wysyłać ramki WebSocket w dowolnej chwili bez nowych żądań HTTP

Implementacja w Kotlin (OkHttp)

```
val client = OkHttpClient.Builder()
    .readTimeout(0, TimeUnit.MILLISECONDS)
    .build()

val request = Request.Builder()
    .url("wss://iot-hub.example.com/ws")
    .build()

val ws = client.newWebSocket(request,
  object : WebSocketListener() {
    override fun onMessage(
        ws: WebSocket, text: String) {
        val data = Json.decodeFromString
            <SensorData>(text)
        _uiState.value = UiState.Data(data)
    }
    override fun onFailure(
        ws: WebSocket, t: Throwable,
        response: Response?) {
        // reconnect logic
    }
})
// Wysyłanie komendy
ws.send("""{"cmd":"setTemp","val":22}""")
```

Bluetooth Low Energy (BLE): łączność krótkiego zasięgu

BLE (Bluetooth 4.0+) jest dominującym protokołem dla IoT krótkiego zasięgu: wearables, czujniki medyczne, smart home.

Model GATT (Generic Attribute Profile)

Profile
Zestaw usług dla danego zastosowania (HR Profile, Battery Service, Custom IoT Profile)

Service
Zestaw powiązanych charakterystyk, identyfikowany przez UUID (np. 0x180D = Heart Rate Service)

Characteristic
Pojedyncza wartość danych, odczyt (Read), zapis (Write), powiadomienia (Notify, Indicate)

Descriptor
Metadane charakterystyki, CCCD (włączenie notyfikacji), User Description, Format

Skanowanie i połączenie (Android)

```
// Skanowanie BLE
val scanner =
bluetoothAdapter.bluetoothLeScanner
val settings = ScanSettings.Builder()
    .setScanMode(SCAN_MODE_LOW_LATENCY).bu
ild()
scanner.startScan(null, settings,
callback)

// Połączenie z urządzeniem
device.connectGatt(context, false, object
    : BluetoothGattCallback() {
  override fun onServicesDiscovered(
      gatt: BluetoothGatt, status: Int) {
    val svc = gatt.getService(SENSOR_UUID)
    val char =
svc.getCharacteristic(TEMP_UUID)

gatt.setCharacteristicNotification(char,tr
ue)
  }
  override fun onCharacteristicChanged(
      gatt: BluetoothGatt,
      char: BluetoothGattCharacteristic) {
    val temp =
char.getIntValue(FORMAT_SINT16,0)
    updateTemperatureUI(temp / 100.0f)
  }
})
```

BLE vs Classic Bluetooth: BLE zużywa ~10× mniej energii, zasięg do 100 m (BLE 5.0), przepustowość do 2 Mbps.

Technologie łączności bezprzewodowej w IoT

Wi-Fi 802.11

Zasięg: 50–100 m
Pasmo: do 9.6 Gbps (Wi-Fi 6)
Energia: Wysokie

Kamery IP, bramy sieciowe, routery IoT, urządzenia strumieniujące wideo

Zigbee (802.15.4)

Zasięg: 10–100 m (mesh)
Pasmo: 250 kbps
Energia: Bardzo niskie

Smart home (żarówki, zamki, czujniki), siatki mesh, Philips Hue, Amazon Echo Plus

Z-Wave

Zasięg: 30 m (max 4 hopy)
Pasmo: 100 kbps
Energia: Niskie

Smart home, mniejsze zakłócenia od Wi-Fi, dedykowany ekosystem (do 232 urządzeń)

Thread (OpenThread)

Zasięg: Mesh, wiele hopów
Pasmo: 250 kbps
Energia: Bardzo niskie

Matter (Apple/Google/Amazon), nowy standard smart home, urządzenia IPv6-native

LoRaWAN

Zasięg: do 15 km
Pasmo: 0.3–50 kbps
Energia: Ekstremalnie niskie

Monitoring infrastruktury, rolnictwo precision, smart city, urządzenia bateryjne na lata

NB-IoT / LTE-M

Zasięg: Sieć komórkowa
Pasmo: 200 kbps / 1 Mbps
Energia: Niskie–średnie

Liczniki mediów, tracking pojazdów, monitoring obszarów bez Wi-Fi

Android Development dla IoT: architektura MVVM

Rekomendowana architektura aplikacji Android IoT oparta na Jetpack i Clean Architecture

UI Layer (View)

- Compose / XML Layouts
- ViewModel (StateFlow)
- Navigation Component
- DataBinding

Domain Layer

- UseCases / Interactors
- Domain Models
- Repository Interfaces
- Business Logic

Data Layer

- Repository Impl.
- Remote (Retrofit/MQTT)
- Local (Room Database)
- WorkManager (background)

IoT Integration

- BLE Manager
- MQTT Client Service
- Sensor Data Parser
- OTA Update Manager

Kluczowe biblioteki Jetpack dla IoT

Hilt	Coroutines	Room	WorkManager	DataStore	CameraX
DI	Async	Lokalna DB	Background	Preferences	AR/Vision

iOS Development dla IoT, Swift i Apple Frameworks

Frameworki Apple dla IoT

HomeKit
Oficjalne API Apple dla smart home, bezpieczna komunikacja z urządzeniami certyfikowanymi przez Apple (MFi)

CoreBluetooth
BLE scanning, pairing, GATT services discovery, pełna kontrola połączeń BLE na poziomie systemu

Network.framework
TLS, TCP/UDP sockets, WebSocket, zastępuje BSD sockets nowoczesnym Swift API

Combine / async-await
Reaktywne strumienie danych z urządzeń IoT, asynchroniczne operacje sieciowe (URLSession async)

Core Data + CloudKit
Lokalne przechowywanie danych telemetrycznych + synchronizacja z iCloud

Przykład CoreBluetooth (Swift)

```
class IoTManager: NSObject,
CBCentralManagerDelegate,
    CBPeripheralDelegate {
  var centralManager: CBCentralManager!
  var peripheral: CBPeripheral?
  let sensorServiceUUID = CBUUID(string: "180D")
  let tempCharUUID = CBUUID(string: "2A1C")

  func centralManagerDidUpdateState(
      _ central: CBCentralManager) {
    if central.state == .poweredOn {
      central.scanForPeripherals(
        withServices: [sensorServiceUUID])
    }
  }
  func peripheral(_ peripheral: CBPeripheral,
    didUpdateValueFor char: CBCharacteristic,
    error: Error?) {
    if let data = char.value {
      let temp = data.withUnsafeBytes {
        $0.load(as: Int16.self)
      }
      DispatchQueue.main.async {
        self.temperature = Double(temp) / 100.0
      }
    }
  }
}
```

Programowanie Cross-Platform dla IoT: Flutter i React Native

Jedna baza kodu dla Android i iOS, kluczowe zalety przy ograniczonych zasobach zespołowych

Flutter (Dart)

- Silnik Skia/Impeller, własne renderowanie
- Hot reload, szybkie iteracje
- flutter_blue_plus (BLE)
- mqtt_client package
- Doskonała wydajność UI
- Rosnące wsparcie IoT ecosystem

React Native (JS/TS)

- Bridge → JSI → Native Modules
- Ogromny ekosystem npm
- react-native-ble-plx
- Łatwiejsze dla web developerów
- Hermes engine (szybki JS)
- Expo, szybki start projektu

Kotlin Multiplatform

- Współdzielona logika biznesowa
- Native UI (Compose/SwiftUI)
- Ktor klient HTTP/WebSocket
- 100% dostęp do natywnych API
- Najlepszy wybór dla krytycznych zastosowań
- Wsparcie JetBrains + Google

Wskazówka: Dla projektu z intensywnym BLE i wymaganą wysoką wydajnością → Flutter lub KMP. Dla szybkich prototypów i dużych zespołów webowych → React Native.

Przetwarzanie danych z czujników w aplikacji mobilnej

Pipeline przetwarzania danych IoT

1. Odbiór

BLE Callback / MQTT onMessage / WebSocket onMessage

2. Parsowanie

Deserializacja JSON/Protocol Buffers/CBOR → obiekt domenowy

3. Walidacja

Zakres wartości, timestamp, integralność danych, outlier detection

4. Filtrowanie

Kalmański, moving average, Savitzky-Golay, redukcja szumu

5. Agregacja

Min/Max/Avg w oknie czasowym, downsampling przed wizualizacją

6. Wizualizacja

Wykres real-time (MPAndroidChart, Charts dla iOS), dashboard

Kluczowe formaty serializacji: JSON (czytelny, duże rozmiary) · Protocol Buffers (kompaktowy, schematowany) · CBOR (binarny JSON, standardowy dla CoAP) · MessagePack (wydajny)

Edge Computing w aplikacjach mobilnych IoT

Przetwarzanie danych bliżej źródła. Na urządzeniu mobilnym lub lokalnej bramie, zamiast w chmurze

Zalety edge computing dla IoT

Niskie opóźnienie

Przetwarzanie lokalne → latencja < 10 ms vs. 50-200 ms dla chmury, krytyczne dla sterowania w czasie rzeczywistym

Praca offline

Aplikacja działa bez połączenia z internetem, lokalna baza danych buforuje dane, synchronizacja przy przywróceniu łącza

Prywatność danych

Wrażliwe dane (np. medyczne) nie opuszczają lokalnej sieci, zgodność z RODO

Oszczędność pasma

Przesyłanie tylko zagregowanych wyników lub anomalii zamiast surowych danych, redukcja kosztów transmisji

ML na urządzeniu (TensorFlow Lite)

```
// Anomaly detection na urządzeniu
val model = TFLiteModel.newInstance(
    context, "anomaly_detector.tflite")

fun detectAnomaly(sensorData: FloatArray)
    : Boolean {
  val input =
TensorBuffer.createFixedSize(
      intArrayOf(1, sensorData.size),
      DataType.FLOAT32)
  input.loadArray(sensorData)

  val outputs = model.process(input)
  val score =
outputs.outputFeature0AsFloatArray

  return score[0] > ANOMALY_THRESHOLD
}
// Zwalniaj zasoby
model.close()
```

Detekcja anomalii

Modele klasyfikacji

Przetwarzanie obrazu

Object detection (YOLO)

NLP na urządzeniu

Rozpoznawanie komend

Platformy Chmurowe dla IoT: AWS, Azure, Google Cloud

AWS IoT Core

- IoT Core, MQTT/HTTP broker
- Greengrass, edge computing
- IoT Analytics, przetwarzanie
- IoT Device Defender, security
- AWS Lambda, serverless

```
SDK: aws-iot-device-sdk-android
```

Azure IoT Hub

- IoT Hub, centralny broker
- IoT Edge, local processing
- Digital Twins, wirtualne modele
- Azure Stream Analytics
- Time Series Insights

```
SDK: azure-iot-sdk-java (Android)
```

Google Cloud IoT

- Cloud IoT Core (→ deprecated)
- Pub/Sub, messaging
- BigQuery, analityka
- Vertex AI, ML pipeline
- Firebase Realtime Database

```
SDK: google-cloud-iot-core + Firebase
```

Kluczowe kryteria wyboru platformy IoT

- Liczba urządzeń i skalowalność
- Koszt (per-message vs. flat fee)
- Geolokalizacja danych (RODO)
- Obsługiwane protokoły
- SLA i dostępność

Integracja aplikacji z chmurą: wzorce i architektura

Direct MQTT Connection

Aplikacja mobilna łączy się bezpośrednio z brokerem MQTT (AWS IoT Core, HiveMQ Cloud). Prosta implementacja, ale wymaga zarządzania połączeniem w tle i obsługi reconnect.

✓ Niskie opóźnienia, real-time

✗ Bateria, zarządzanie połączeniem

REST + WebSocket Hybrid

REST API dla operacji CRUD i historycznych danych. WebSocket lub SSE (Server-Sent Events) dla live updates. Łatwe cachowanie przez CDN.

✓ Cacheable, skalowalny

✗ Wyższe opóźnienia niż MQTT

Firebase Realtime / Firestore

Firebase jako pośrednik, Cloud Functions reagują na dane IoT w MQTT i zapisują do Firestore. Aplikacja używa Firebase SDK z lokalnym cache offline.

✓ Offline first, prosty SDK

✗ Vendor lock-in Google

Wybór wzorca zależy od wymagań:
latencja krytyczna → MQTT Direct | łatwy start → Firebase | skalowalność enterprise → REST+WS

Zagrożenia i ochrona aplikacji mobilnych IoT

"Security by Design", bezpieczeństwo wbudowane od początku projektu, nie dodane na końcu

Zagrożenia

Man-in-the-Middle
Przechwycenie niezaszyfrowanej komunikacji MQTT/HTTP między app, a brokerem

Device Spoofing
Fałszywe urządzenie podszywające się pod legalny sensor w sieci IoT

Replay Attack
Ponowne wysłanie przechwyconych pakietów sterowania urządzeniem

Insecure Storage
Tokeny i certyfikaty w logach, SharedPreferences bez szyfrowania

OTA Poisoning
Podmiana firmware podczas aktualizacji Over-the-Air

✓ Środki ochrony

TLS 1.3 + MQTT over TLS
Szyfrowanie E2E, weryfikacja certyfikatu serwera (certificate pinning)

Mutual TLS (mTLS)
Certyfikaty X.509 dla każdego urządzenia, obustronnieuwe uwierzytelnienie

Timestamp + Nonce
Każda wiadomość zawiera timestamp i jednorazowy nonce, blokuje replay

Android Keystore / iOS Secure Enclave
Przechowywanie kluczy w hardware, niedostępne dla innych aplikacji

Podpis cyfrowy firmware
Weryfikacja podpisu ECDSA przed instalacją aktualizacji

Autentykacja i autoryzacja w systemach IoT

Metody uwierzytelniania

Username/Password + TLS
Podstawowa metoda, niewystarczająca dla produkcji. Hasła muszą być silne i przechowywane jako hash (bcrypt).

API Key / Token
Statyczny klucz przypisany do urządzenia. Prosty, ale bez expiry i trudny do rotacji w dużych flotach.

JWT (JSON Web Tokens)
Tokeny z expiry, podpisane kluczem (RS256/ES256). Aplikacja mobilna przechowuje refresh token, wymienia na access token.

OAuth 2.0 / OIDC
Delegacja uprawnień, użytkownik autoryzuje apkę do dostępu do swoich urządzeń bez podawania hasła.

mTLS + X.509 Certs
Certyfikaty per urządzenie, gold standard dla produkcji IoT. AWS IoT Core, Azure IoT Hub.

JWT Flow w aplikacji mobilnej

1. Login (email/pwd) → Auth Server
2. Serwer zwraca access_token (1h) + refresh_token (30d)
3. App dołącza Bearer token do każdego żądania HTTP / MQTT username
4. Po wygaśnięciu: refresh_token → nowy access_token
5. Refresh token rotacja, zmniejsza ryzyko kradzieży

Role-Based Access Control (RBAC)

Admin	Operator	Viewer
Pełny dostęp + zarządzanie flotą	Sterowanie i monitoring	Tylko odczyt danych

Persystencja danych i synchronizacja offline-first

Aplikacje IoT muszą działać przy słabym lub braku połączenia, strategia offline-first jest kluczowa

Lokalne bazy danych

Room (Android)

SQLite wrapper z walidacją schema w compile-time, obsługa Flows dla reaktywnych zapytań, migracje

Core Data (iOS)

Managed Object Context, persistent stores, NSFetchedResultsController do reaktywnych list

SQLite.swift / GRDB

Cross-platform (KMP), typesafety, FTS (full-text search) dla historii danych IoT

Realm Database

Mobilna baza obiektowa, szybka, reaktywna, z synchronizacją Atlas Sync (MongoDB)

Strategia synchronizacji

```
// Offline queue pattern (Kotlin)
class IoTDataRepository(
    private val localDb: SensorDao,
    private val remoteApi: IoTApiService,
    private val networkMonitor: NetworkMonitor
) {
  fun saveMeasurement(data: SensorData) {
    localDb.insert(data.copy(synced = false))
    if (networkMonitor.isConnected) {
      syncPending()
    }
  }

  private fun syncPending() {
    val unsynced = localDb.getUnsynced()
    unsynced.forEach { data ->
      remoteApi.upload(data).onSuccess {
        localDb.markSynced(data.id)
      }
    }
  }
}
```

Wzorce konfliktu synchronizacji

Last-Write-Wins
Prosta, może tracić dane

CRDT
Bez konfliktów, złożona

Server-Authoritative
Serwer rozstrzyga

Przetwarzanie w tle i powiadomienia push w IoT

Android: Background Processing

Foreground Service
MQTT connection utrzymywana przez Foreground Service, widoczna notyfikacja wymagana przez OS. Najlepsza opcja dla ciągłego IoT.

WorkManager
Zadania periodyczne (synchronizacja, raportowanie). Gwarancja wykonania nawet po restarcie. Obsługuje Doze Mode.

AlarmManager
Dokładne timingi (np. próbkowanie co 1 min). Wymaga pozwolenia SCHEDULE_EXACT_ALARM (Android 12+).

JobScheduler
Elastyczne warunki (sieć, ładowanie). Wbudowany w system, nie wymaga Wake Locks.

IOS: Background Modes

Background Fetch
System wywołuje apkę co jakiś czas (≥15 min), ograniczone, system decyduje o częstotliwości

Push to Sync (Silent Push)
Serwer wysyła APNs silent notification → apka budzi się i synchronizuje (30s limit)

Background URLSession
Pobieranie/wysyłanie dużych danych w tle, transfer kontynuowany po zawieszeniu apki

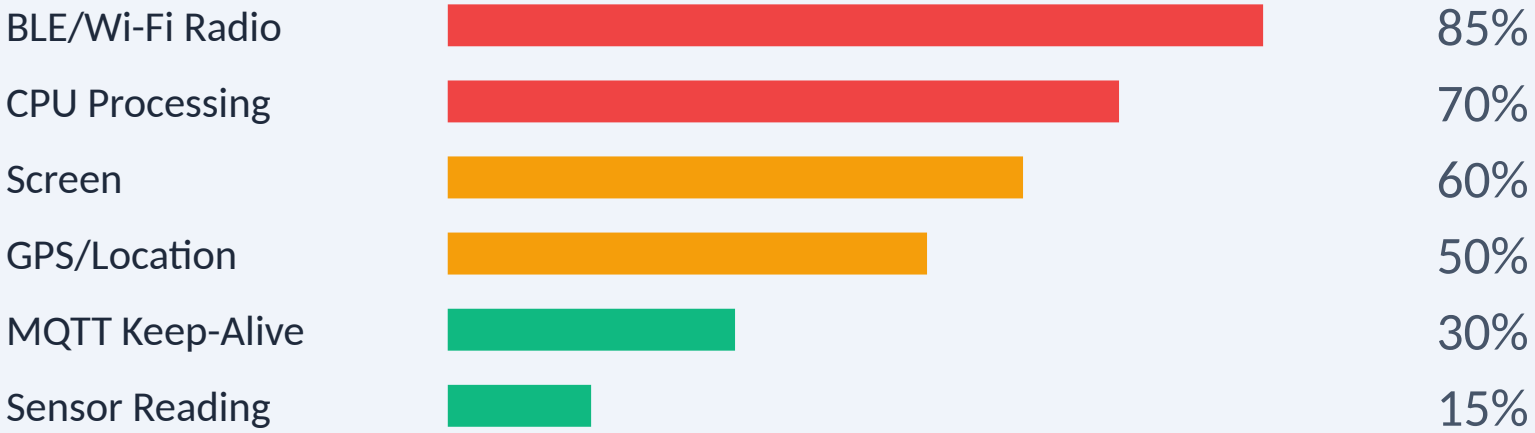
CoreBluetooth CBManager
Stan.restoration, odtworzenie połączeń BLE po restarcie apki, stała łączność z urządzeniami

Push Notifications dla IoT, FCM (Android) / APNs (iOS): alerty progowe, alarmy bezpieczeństwa, statusy urządzeń, raporty agregowane

Optymalizacja zużycia energii w aplikacjach IoT

Urządzenia IoT są często zasilane bateryjnie, optymalizacja energetyczna jest wymaganiem niefunkcjonalnym pierwszej klasy

Główne konsumenci energii



Techniki optymalizacji

- Adaptive polling, rzadsze próbkowanie gdy dane stabilne
- Batch uploads, grupowanie danych i wysyłka rzadko
- BLE connection interval tuning, dłuższe interwały przy nieaktywności
- Data compression przed transmisją, mniejsze pakiety

Projektowanie UX/UI dla aplikacji mobilnych IoT

Dobre UI/UX dla IoT musi priorytetyzować szybkość dostępu do informacji krytycznych i intuicyjność sterowania

① Hierarchia informacji

Krytyczne alerty na pierwszym planie. Dashboard ze statusem wszystkich urządzeń na głównym ekranie. Szczegóły na 2. poziomie nawigacji.

② Status w czasie rzeczywistym

Wskaźniki online/offline dla każdego urządzenia. Timestamp ostatniego odczytu. Animowane zmiany wartości (nie tylko suche liczby).

③ Kontrolki dotykowe

Przyciski sterowania minimum 44×44dp (Android) / 44pt (iOS). Toggle switches zamiast pól tekstowych dla stanu ON/OFF.

④ Obsługa błędów

Wyraźna informacja o utracie połączenia. Pokazuj ostatnią znana wartość z timestampem. Retry mechcanism z exponential backoff.

⑤ Powiadomienia kontekstowe

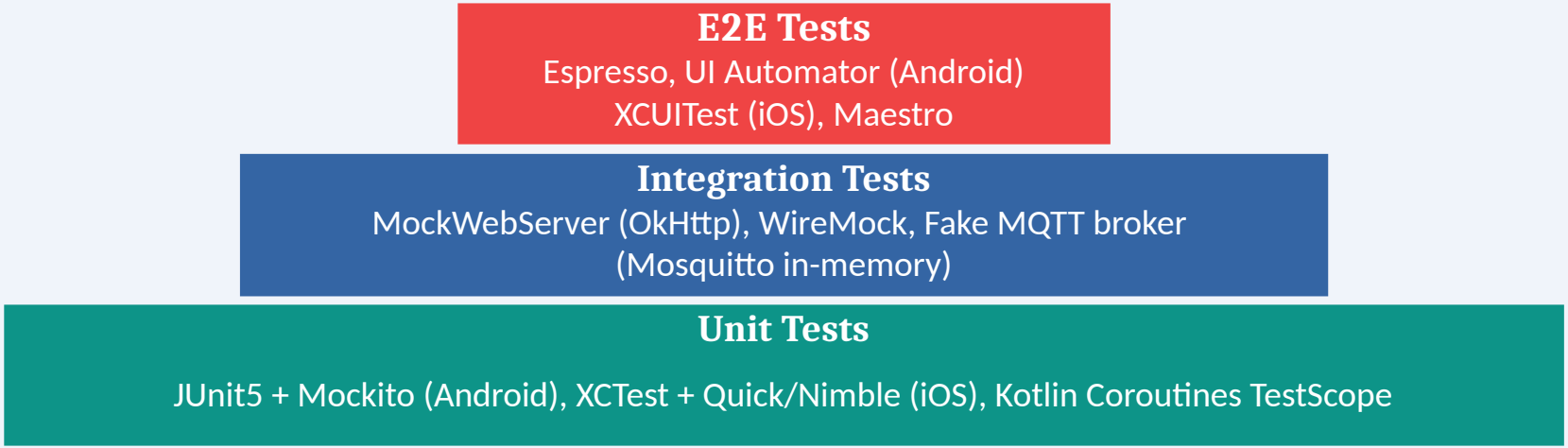
Alerty z kontekstem, nie tylko 'Alarm!' ale 'Temperatura w salonie: 38°C (próg: 30°C), 14:32'. Możliwość wyciszenia z ekranu powiadomienia.

⑥ Dark Mode i dostępność

Dark Mode ważny dla urządzeń przemysłowych (słaby kontrast w jasnych warunkach). Wsparcie TalkBack/VoiceOver, minimalny kontrast 4.5:1.

Strategie i narzędzia testowania aplikacji IoT

Piramida testów IoT: testy jednostkowe dominują, integracyjne pokrywają protokoły, E2E walidują scenariusze użytkownika



Specyfika testowania IoT

Symulacja urządzeń

Wirtualny sensor w Dockerze z MQTT, generowanie danych testowych

Testy połączenia

Symulacja utraty sieci, wolnego łącza (Charles Proxy, tc netem)

Testy protokołów

Walidacja poprawności wiadomości MQTT/CoAP, sprawdzanie QoS

Testy bezpieczeństwa

Penetration testing (OWASP Mobile), Zed Attack Proxy (ZAP)

Zarządzanie flotą urządzeń IoT z aplikacji mobilnej

Device Management obejmuje cały lifecycle urządzenia, od provisioning po decommissioning

Provisioning → Rejestracja → Monitoring → Aktualizacja (OTA) → Diagnostyka → Decommission

Provisioning (QR Code / BLE)

Użytkownik skanuje QR kod na urządzeniu → aplikacja wysyła dane Wi-Fi i certyfikat X.509 przez BLE (secure channel) → urządzenie łączy się z brokerem IoT. Użycie: AWS IoT Just-in-Time-Provisioning.

Over-The-Air (OTA) Update

Aplikacja mobilna pobiera manifest aktualizacji z serwera (wersja, URL, hash SHA-256 firmware). Weryfikacja podpisu cyfrowego. Wysłanie URL przez MQTT do urządzenia. Monitorowanie postępu w czasie rzeczywistym.

Remote Diagnostics

Shadow Device / Device Twin, wirtualna kopia stanu urządzenia w chmurze. Różnica między stanem reported (rzeczywistym) a desired (oczekiwanym) uruchamia akcję naprawczą. Logi urządzeń przez MQTT diagnostics topic.

Fleet Analytics

Dashboard z agregowanymi statystykami: dostępność urządzeń (%), średnia latencja, liczba błędów, zużycie baterii floty. Alerty gdy > X% urządzeń offline. Eksport danych CSV/PDF z aplikacji.

Case Study 1: Aplikacja Smart Home z IoT

Architektura systemu

Urządzenia

Czujniki temp/wilg (Zigbee), żarówki Hue, zamek Z-Wave, kamera IP, termostat Z-Wave

Hub lokalny

Raspberry Pi 4 + Home Assistant + Mosquitto MQTT Broker. Brama do chmury przez VPN.

Chmura

AWS IoT Core jako MQTT broker w chmurze, DynamoDB (historia), Lambda (automatyzacje), SNS (alerty)

Aplikacja mobilna

Flutter, Android + iOS z jednej bazy kodu. Material You design. Offline support z Hive DB.

Funkcjonalności i wyzwania

Zaimplementowane funkcje

- Dashboard z wszystkimi urządzeniami (live status)
- Automatyzacje (jeśli temp > 25°C → włącz AC)
- Historia danych, wykresy 24h / 7d / 30d
- Geofencing, auto zamykanie zamka po wyjeździe
- Widgety ekranu głównego (Android 12+)

Napotkane wyzwania

Heterogeniczność protokołów:
Rozwiązanie: Home Assistant jako universal translator

Latencja przez VPN:
Rozwiązanie: lokalne MDNS discovery + direct LAN fallback

Zarządzanie baterią:
Rozwiązanie: adaptacyjne interwały polling + push only dla alertów

Case Study 2: Przemysłowy system monitoringu maszyn (IIoT)

System monitoringu parku maszynowego fabryki - 500 maszyn, 15 000 czujników, 200 TB danych/rok

<5 ms	99.99%	40%	18 mies.
Latencja alertów	Dostępność systemu	Redukcja awarii	ROI projektu

Stos technologiczny

- Czujniki → PLC → OPC-UA → MQTT Gateway → AWS IoT Core
- TimescaleDB (PostgreSQL + time-series) dla danych historycznych
- Apache Kafka dla event streaming (anomalie, alerty)
- TensorFlow Lite, predictive maintenance modele na tablet Android
- React Native tablet app, operator panel z AR overlay (kamera)

Kluczowe wnioski z projektu

Protocol translation krytyczny
OPC-UA, Modbus, PROFINET - każdy PLC mówi innym językiem. Gateway jest kluczowy.

Edge ML opłacalne
Detekcja anomalii na urządzeniu zmniejsza transmisję do chmury o 95%. Koszt: model 2 MB.

Security compliance
ISO 27001 + IEC 62443; regulatorzy przemysłowi wymagają formalnego audytu bezpieczeństwa.

Crew training kluczowy
Najlepszy system nie zadziała bez przeszkolonego personelu. 3-miesięczny plan wdrożenia.

Podsumowanie

Matter Standard
Nowy unified protocol (Apple/Google/Amazon/Samsung), jeden ekosystem smart home

AI/ML na urządzeniu
TFLite, CoreML, LiteRT, coraz potężniejsze modele działające offline na mobile

5G + NB-IoT
Ultra-niska latencja 5G dla krytycznych zastosowań, NB-IoT dla masowych deploymentów

Digital Twins
Wirtualne repliki urządzeń fizycznych, symulacja, predykcja awarii, optymalizacja

Edge AI Chips
Apple Neural Engine, Google Edge TPU, ARM Ethos, dedykowany hardware AI na urządzeniach

WebAssembly IoT
WASM jako universal runtime, jedna logika biznesowa na mikrokontrolery, mobile i cloud

Kluczowe wnioski wykładu

- MQTT + TLS = fundament bezpiecznej komunikacji IoT dla aplikacji mobilnych
- Architektura MVVM/Clean Architecture z offline-first to standard produkcyjny
- Edge computing i ML on-device zmniejszają latencję i koszty transmisji
- Security by Design, bezpieczeństwo od pierwszego dnia projektu, nie na końcu

Pytania, dyskusja i koniec