PROJEKT APLIKACJI MOBILNEJ

SensorHub – Platforma Edukacyjna

Obsługa Sensorów i Technologii Mobilnych w Android Studio

Wersja dokumentu: 1.0

Data utworzenia: Luty 2026

Cel projektu:

Stworzenie kompleksowej aplikacji mobilnej demonstrującej praktyczne wykorzystanie sensorów urządzenia mobilnego, zaawansowanych interfejsów użytkownika oraz technik przetwarzania afektywnego dla celów edukacyjnych.

Spis treści

[Spis treści zostanie automatycznie wygenerowany w Microsoft Word: kliknij tutaj i wybierz Odwołania > Spis treści > Automatyczny spis treści]

Spis treści

1. Wprowadzenie

1.1. Cel projektu

Projekt SensorHub ma na celu stworzenie wszechstronnej aplikacji edukacyjnej dla studentów kierunków informatycznych i inżynierskich, która w przystępny sposób prezentuje zaawansowane możliwości współczesnych urządzeń mobilnych. Aplikacja stanowi kompleksowe narzędzie demonstracyjne, które łączy teorię z praktyką, umożliwiając studentom zrozumienie i eksperymentowanie z różnymi aspektami programowania mobilnego.

1.2. Zakres funkcjonalny

Aplikacja obejmuje następujące główne obszary funkcjonalne:

- Obsługa sensorów urządzenia mobilnego (akcelerometr, żyroskop, magnetometr, światło, GPS, etc.)
- Różnorodne interfejsy interakcji z użytkownikiem (dotyk, gesty, głos, wibr)
- Zaawansowane projektowanie UI/UX z wykorzystaniem Material Design 3
- Grafika i animacje (Canvas API, animacje property, transitions)
- Przetwarzanie afektywne - analiza stanu emocjonalnego użytkownika na podstawie danych sensorycznych
- Wizualizacja danych w czasie rzeczywistym

1.3. Technologie i narzędzia

Kategoria	Technologia/Narzędzie
Język programowania	Kotlin (preferowany) / Java
IDE	Android Studio (najnowsza stabilna wersja)
Minimalna wersja Android	Android 8.0 (API 26) – docelowa: Android 14 (API 34)
Architektura	MVVM (Model-View-ViewModel)
UI Framework	Jetpack Compose + Material Design 3
Kluczowe biblioteki	Lifecycle, ViewModel, LiveData, Navigation, Room, WorkManager, Hilt/Dagger

2. Architektura aplikacji

2.1. Wzorzec MVVM

Aplikacja wykorzystuje architekturę MVVM (Model-View-ViewModel), która zapewnia czyste rozdzielenie logiki biznesowej od warstwy prezentacji. Taki podział ułatwia testowanie, utrzymanie i rozwój aplikacji.

Warstwy aplikacji:

- **Model** – klasy danych, repozytoria, źródła danych (sensory, baza danych Room)
- **ViewModel** – zarządza stanem UI, przetwarza dane z modelu, obsługuje logikę biznesową
- **View** – komponenty Jetpack Compose wyświetlające interfejs użytkownika

2.2. Struktura pakietów

`com.example.sensorhub/`

- **`ui/`** – komponenty interfejsu użytkownika
  - `screens/` – ekrany aplikacji
  - `components/` – komponenty wielokrotnego użytku
  - `theme/` – motyw Material Design
- **`viewmodel/`** – ViewModele dla poszczególnych funkcji
- **`data/`** – warstwa danych
  - `model/` – klasy modeli danych
  - `repository/` – repozytoria
  - `database/` – baza danych Room
- **`sensors/`** – menedżery sensorów
- **`utils/`** – klasy pomocnicze i rozszerzenia
- **`affective/`** – moduł przetwarzania afektywnego

3. Moduł obsługi sensorów

3.1. Przegląd sensorów

Moduł sensorów stanowi rdzeń aplikacji, prezentując różnorodne czujniki dostępne w urządzeniach mobilnych. Każdy sensor ma dedykowany ekran z wizualizacją danych w czasie rzeczywistym oraz szczegółowym opisem działania.

Sensor	Typ danych	Zastosowania edukacyjne
Akcelerometr	Przyspieszenie (x, y, z) w m/s²	Detekcja ruchu, potrząsanie, orientacja urządzenia
Żyroskop	Prędkość kątowa (x, y, z) w rad/s	Rotacja urządzenia, gry VR, stabilizacja kamery
Magnetometr	Pole magnetyczne (x, y, z) w µT	Kompas, nawigacja, detekcja metalów
Światło	Natężenie światła w lux	Automatyczna jasność ekranu, fotografia
GPS	Szerokość i długość geograficzna, wysokość, prędkość	Nawigacja, śledzenie aktywności, geolokalizacja

Zbliżeniowy	Odległość obiektu w cm	Wykrywanie obecności przy ekranie, automatyczne wyłączanie ekranu
Barometr	Ciśnienie atmosferyczne w hPa	Określanie wysokości, prognoza pogody

3.2. Implementacja menedżera sensorów

Przykładowa klasa SensorManager dla akcelerometru:

```
class AccelerometerManager(context: Context) {private val sensorManager = context.getSystemService(SENSOR_SERVICE) as SensorManager private val accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) fun startListening(onDataChanged: (x: Float, y: Float, z: Float) -> Unit) {sensorManager.registerListener( object : SensorEventListener { override fun onSensorChanged(event: SensorEvent) { onDataChanged(event.values[0], event.values[1], event.values[2]) } override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {} }, accelerometer, SensorManager.SENSOR_DELAY_UI ) } }
```

4. Interfejsy interakcji z użytkownikiem

4.1. Obsługa gestów dotykowych

Aplikacja demonstruje różne rodzaje gestów wspierane przez Jetpack Compose:

- **Pojedyncze dotknięcie (Tap)** – podstawowa interakcja
- **Długie przytrzymanie (Long Press)** – wyświetlanie menu kontekstowego
- **Przeciąganie (Drag)** – przesuwanie obiektów
- **Przesuwanie (Swipe)** – nawigacja między ekranami
- **Uszczypnięcie (Pinch)** – zoom in/out
- **Obrót (Rotate)** – rotacja elementów

4.2. Rozpoznawanie mowy

Moduł rozpoznawania mowy wykorzystuje Android Speech Recognizer API do konwersji mowy na tekst. Aplikacja prezentuje praktyczne zastosowania.

- Sterowanie głosowe aplikacją
- Dyktowanie notatek
- Wyszukiwanie głosowe
- Wykrywanie komend głosowych (np. start, stop, reset).

4.3. Wibracje i haptyka

Aplikacja demonstruje różne wzorce wibracji jako formę feedbacku haptycznego: Krótka wibracja potwierdzenia (50ms) Dłuższa wibracja ostrzeżenia (200ms) Wzorce wibracji (morse code) Adaptacyjna haptyka zależna od intensywności akcji

5. Projektowanie UI/UX

5.1. Material Design 3

Aplikacja w pełni wykorzystuje Material Design 3 (Material You), oferując: Dynamiczne kolorowanie (Dynamic Color) – dostosowanie palety kolorów do tapety użytkownika Adaptacyjne komponenty – przyciski, karty, nawigacja dostosowane do kontekstu Ulepszone animacje i przejścia Zarządzanie motywami jasnym i ciemnym

5.2. Nawigacja

Struktura nawigacji:

- **Bottom Navigation Bar** – główne sekcje aplikacji (Sensory, Interakcje, Afektywność, Ustawienia)
- **Navigation Drawer** – dodatkowe opcje i informacje
- **Hierarchiczna nawigacja** – logiczne grupowanie funkcji
- **Breadcrumbs** – ścieżka nawigacji dla zagnieżdżonych ekranów

5.3. Responsywność i dostępność

- Adaptacyjny layout dla różnych rozmiarów ekranów (telefony, tablety, foldables)
- Wsparcie dla orientacji pionowej i poziomej
- Etykiety dostępności (Content Descriptions)
- Wsparcie TalkBack dla użytkowników niewidomych
- Skalowanie czcionek zgodnie z ustawieniami systemowymi
- Wysoki kontrast i czytelność kolorów (WCAG 2.1 AA)

6. Grafika i animacje

6.1. Canvas API

Moduł graficzny wykorzystuje Canvas do niestandardowych wizualizacji danych z sensorów.

- Wykres 3D akcelerometru w czasie rzeczywistym
- Kompas magnetyczny z animowaną igłą
- Wizualizacja poziomu dźwięku (audio waveform)
- Efekty cząsteczkowe (particle effects) reagujące na ruchy

6.2. Animacje w Compose

Rodzaje animacji:

- **animateFloatAsState** – proste animacje wartości liczbowych
- **animateColorAsState** – płynne przejścia kolorów
- **AnimatedVisibility** – animowane pokazywanie/ukrywanie elementów
- **Crossfade** – płynne przełączanie między widokami
- **updateTransition** – złożone animacje wieloetapowe
- **Spring animations** – naturalne animacje z efektem sprężyny

6.3. Przykłady zastosowań

- Animowana prezentacja danych sensorycznych z użyciem wykresów liniowych i kołowych
- Płynne przejścia między ekranami z efektami shared element transition
- Animowane wskaźniki postępu i ładowania
- Mikro-interakcje: feedback wizualny dla akcji użytkownika (ripple effect, elevation changes)

7. Moduł przetwarzania afektywnego

7.1. Koncepcja

Przetwarzanie afektywne (affective computing) polega na rozpoznawaniu, interpretacji i symulowaniu ludzkich emocji. W kontekście aplikacji mobilnej wykorzystujemy dane z sensorów do wnioskowania o stanie emocjonalnym użytkownika.

7.2. Źródła danych afektywnych

Źródło danych	Korelacja ze stanem emocjonalnym
Wzorce ruchu (akcelerometr)	Gwałtowne, chaotyczne ruchy mogą wskazywać na stres lub frustrację. Płynne, spokojne ruchy sugerują relaksację.
Częstotliwość interakcji dotykowych	Szybkie, impulsywne dotknięcia mogą wskazywać na niecierpliwość lub nerwowość. Powolne, dokładne - na spokój i skupienie.
Siła nacisku ekranu	Zwiększony nacisk może wskazywać na napięcie emocjonalne lub determinację.
Czas spędzony w aplikacji	Długie sesje mogą wskazywać na zaangażowanie pozytywne lub prokrastynację. Krótkie - na brak zainteresowania.
Ton głosu (rozpoznawanie mowy)	Wysokość, tempo i modulacja głosu mogą wskazywać na różne stany emocjonalne (radość, smutek, złość).

7.3. Algorytmy klasyfikacji

Aplikacja wykorzystuje proste algorytmy uczenia maszynowego do klasyfikacji stanów emocjonalnych.

- **Detekcja anomalii** – wykrywanie nietypowych wzorców zachowania
- **Analiza trendów** – śledzenie zmian w czasie (poprawa/pogorszenie nastroju)
- **Klasyfikacja wieloklasowa** – przypisanie do kategorii emocjonalnych (radość, smutek, neutralny, stres)
- **Model regresji** – określenie intensywności emocji (skala 0-10)

7.4. Zastosowania edukacyjne

- Zrozumienie podstaw uczenia maszynowego i przetwarzania danych
- Eksploracja etycznych aspektów monitorowania emocji
- Praktyczne zastosowanie analizy danych w czasie rzeczywistym
- Integracja różnych źródeł danych (sensor fusion)
- Podstawy psychologii afektywnej i Human-Computer Interaction (HCI)

8. Wizualizacja danych

8.1. Wykresy i grafy

Aplikacja wykorzystuje bibliotekę MPAndroidChart (lub podobną) do tworzenia interaktywnych wykresów.

- **Wykresy liniowe** – śledzenie zmian wartości sensorów w czasie
- **Wykresy słupkowe** – porównanie wartości z różnych sensorów
- **Wykresy kołowe** – procentowy rozkład danych
- **Wykresy radarowe** – wizualizacja wielowymiarowych danych sensorycznych
- **Heat mapy** – gęstość aktywności użytkownika

8.2. Dashboard w czasie rzeczywistym

Główny ekran aplikacji prezentuje dashboard ze wszystkimi aktywnymi sensorami.

- Automatyczne odświeżanie co 100ms,
- Przełączanie między różnymi widokami (lista/siatka),
- Filtrowanie i sortowanie danych,
- Eksport danych do CSV/JSON.

9. Baza danych i perzystencja

9.1. Room Database

Aplikacja wykorzystuje Room do lokalnego przechowywania danych.

- Historia odczytów sensorów (timestamp, typ sensora, wartości)
- Zapisane sesje pomiarowe
- Preferencje użytkownika
- Dane treningu modelu afektywnego

9.2. DataStore

Preferences DataStore służy do przechowywania ustawień aplikacji:

- Motyw (jasny/ciemny/automatyczny)
- Częstotliwość odświeżania danych
- Włączone/wyłączone sensory
- Język aplikacji
- Preferencje dotyczące prywatności

10. Uprawnienia i bezpieczeństwo

10.1. Wymagane uprawnienia

AndroidManifest.xml:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.RECORD_AUDIO" /> <uses-
permission android:name="android.permission.VIBRATE" /> <uses-permission
android:name="android.permission.BODY_SENSORS" /> <uses-feature
android:name="android.hardware.sensor.accelerometer" /> <uses-feature
android:name="android.hardware.sensor.gyroscope" />
```

10.2. Runtime Permissions

Aplikacja implementuje obsługę uprawnień runtime zgodnie z najlepszymi praktykami Android: Sprawdzanie uprawnień przed dostępem do zasobów Wyjaśnianie użytkownikowi, dlaczego uprawnienie jest potrzebne Graceful degradation - aplikacja działa z ograniczoną funkcjonalnością bez uprawnień Obsługa przypadków, gdy użytkownik odmówi uprawnienia

10.3. Prywatność danych

- Wszystkie dane są przechowywane lokalnie na urządzeniu
- Brak wysyłania danych do zewnętrznych serwerów
- Opcja usunięcia wszystkich zgromadzonych danych
- Przejrzystość - informowanie użytkownika, jakie dane są zbierane i w jakim celu

11. Strategia testowania

11.1. Testy jednostkowe

Testy jednostkowe dla logiki biznesowej i ViewModeli:

- JUnit 4/5 dla podstawowych testów
- MockK dla mockowania zależności
- Turbine dla testowania Flow
- Pokrycie kodu na poziomie min. 70%

11.2. Testy UI

- Jetpack Compose Testing dla testowania komponentów UI
- Espresso dla testów integracyjnych
- Screenshot testing dla weryfikacji wizualnej

11.3. Testy wydajności

- Benchmarking częstotliwości odczytu sensorów
- Profilowanie zużycia baterii
- Analiza memory leaks (LeakCanary)

12. Plan implementacji

12.1. Faza 1: Fundament

- Konfiguracja projektu w Android Studio
- Implementacja architektury MVVM
- Konfiguracja Jetpack Compose i Material Design 3
- Podstawowa nawigacja między ekranami
- Implementacja bazy danych Room

12.2. Faza 2: Moduł sensorów

- Implementacja menedżerów sensorów (akcelerometr, żyroskop, magnetometr)
- Ekrany wizualizacji danych z poszczególnych sensorów
- Implementacja pozostałych sensorów (GPS, światło, zbliżeniowy)
- Dashboard czasu rzeczywistego
- Zapisywanie danych do bazy

12.3. Faza 3: Interfejsy interakcji

- Implementacja obsługi gestów dotykowych
- Rozpoznawanie mowy (Speech-to-Text)
- Wibracje i feedback haptyczny
- Interaktywne demonstracje

12.4. Faza 4: Grafika i animacje

- Implementacja Canvas dla niestandardowych wizualizacji
- Animacje Compose (transitions, spring animations)
- Interaktywne wykresy (MPAndroidChart)
- Efekty wizualne i mikro-interakcje

12.5. Faza 5: Przetwarzanie afektywne

- Zbieranie i normalizacja danych z różnych źródeł
- Implementacja algorytmów klasyfikacji
- Trenowanie i walidacja modeli
- Wizualizacja wyników analizy afektywnej
- Raporty i statystyki

12.6. Faza 6: Finalizacja

- Testy i debugowanie
- Optymalizacja wydajności
- Dokumentacja kodu i materiały edukacyjne
- Przygotowanie wersji demonstracyjnej
- Publikacja w Google Play (opcjonalnie)

13. Przykładowa struktura kodu

13.1. ViewModel dla akcelerometru

```
class AccelerometerViewModel(private val repository: SensorRepository) : ViewModel() {
private val _sensorData = MutableStateFlow(AccelerometerData())     val sensorData:
StateFlow<AccelerometerData> = _sensorData.asStateFlow()          fun
startMonitoring() {         viewModelScope.launch
{             repository.getAccelerometerFlow()                 .collect { data ->
_sensorData.value = data                      repository.saveSensorReading(data)
}         }     }          fun stopMonitoring()
{         repository.stopAccelerometer()     } }
```

13.2. Ekran Compose

```
@Composable fun AccelerometerScreen(viewModel: AccelerometerViewModel) {     val
sensorData by viewModel.sensorData.collectAsState()          Column(modifier =
Modifier.fillMaxSize().padding(16.dp)) {         Text("Akcelerometr", style =
MaterialTheme.typography.headlineMedium)         Spacer(modifier =
Modifier.height(16.dp))                  SensorCard(label = "X", value = sensorData.x,
unit = "m/s²")         SensorCard(label = "Y", value = sensorData.y, unit = "m/s²")
SensorCard(label = "Z", value = sensorData.z, unit = "m/s²")
AccelerometerVisualization(data = sensorData)                  Button(onClick =
{ viewModel.startMonitoring() }) {             Text("Start")         }     } }
```

14. Materiały edukacyjne dla studentów

14.1. Dokumentacja wewnętrzna

Każdy moduł aplikacji zawiera szczegółową dokumentację w formie komentarzy KDoc:

- Opis działania każdej klasy i funkcji
- Parametry i zwracane wartości
- Przykłady użycia
- Odniesienia do dokumentacji Android

14.2. Ćwiczenia praktyczne

Repozytorium projektu zawiera zestaw zadań dla studentów:

- Dodanie nowego sensora (krok po kroku)
- Implementacja własnej wizualizacji danych
- Modyfikacja algorytmu przetwarzania afektywnego
- Stworzenie nowej animacji lub efektu wizualnego
- Optymalizacja wydajności wybranego modułu

14.3. Prezentacje i tutoriale

- Slajdy prezentacyjne dla każdego modułu
- Video-tutoriale demonstrujące kluczowe koncepcje
- Interaktywne demo w samej aplikacji (tryb edukacyjny)
- Quizy i testy wiedzy

15. Podsumowanie i kolejne kroki

15.1. Główne osiągnięcia projektu

Projekt aplikacji SensorHub stanowi kompleksowe narzędzie edukacyjne, które łączy teorię z praktyką w następujących obszarach:

- **Programowanie mobilne** – wykorzystanie nowoczesnych technologii Android (Kotlin, Jetpack Compose, MVVM)
- **Systemy embedded** – bezpośrednia interakcja z sensorami urządzenia
- **UI/UX Design** – projektowanie intuicyjnych i estetycznych interfejsów
- **Uczenie maszynowe** – podstawy klasyfikacji i przetwarzania afektywnego
- **Wizualizacja danych** – prezentacja danych w czasie rzeczywistym
- **Software Engineering** – architektura, testowanie, dokumentacja

15.2. Możliwości rozszerzenia

Aplikacja może być rozwijana w następujących kierunkach:

- Integracja z zewnętrznymi API (pogoda, aktywność fizyczna)
- Synchronizacja danych między urządzeniami (Firebase)
- Zaawansowane modele ML (TensorFlow Lite)
- Rozpoznawanie aktywności (chodzenie, bieganie, jazda samochodem)
- Integracja z urządzeniami wearable (smartwatch)
- Gamifikacja - wyzwania i osiągnięcia

15.3. Wsparcie i zasoby

- **Repozytorium GitHub:** github.com/PRZ-KIA/SensorHub

Dziękujemy za zainteresowanie projektem SensorHub! Zapraszamy do eksploracji fascynującego świata technologii mobilnych.

Załączniki

A. Słownik pojęć

- **MVVM** - Model-View-ViewModel, wzorzec architektoniczny oddzielający logikę biznesową od UI
- **Jetpack Compose** - nowoczesny toolkit do budowania interfejsów użytkownika w Android
- **Room** - biblioteka ORM do lokalnej bazy danych SQLite
- **Flow** - asynchroniczny strumień danych w Kotlin
- **Affective Computing** - przetwarzanie afektywne, rozpoznawanie i symulowanie emocji
- **Sensor Fusion** - łączenie danych z wielu sensorów dla lepszej dokładności

B. Źródła i bibliografia

1. Android Developers Documentation - developer.android.com
2. Kotlin Programming Language - kotlinlang.org
3. Material Design 3 - m3.material.io
4. Jetpack Compose - developer.android.com/jetpack/compose
5. Affective Computing - Rosalind Picard, MIT Press