🛡️ Technischer Entwurf: PocketGuardian

1. Funktionale Anforderungen (messbar)

ID	Anforderung	Beschreibung
F-01	Bewegungserkennung aktivieren	Benutzer kann Bewegungsüberwachung aktivieren/deaktivieren
F-02	Notfallkontakt hinzufügen	Benutzer kann Kontakte mit E-Mail und Telefonnummer speichern
F-03	Kamera automatisch auslösen	Bei Bewegung löst App automatisch Dual-Kamera-Aufnahme aus
F-04	Notfall-E-Mail versenden	Nach Countdown wird automatisch E-Mail mit Bildern versendet
F-05	Countdown starten & abbrechen	Countdown mit Abbruchfunktion startet nach erkannter Bewegung
F-06	Pocket-Erkennung aktivieren	Auto-Modus erkennt, ob sich das Gerät in der Hosentasche befindet
F-07	Testfunktionen nutzen	Benutzer kann Sensoren und Kamera manuell testen

2. Nicht-funktionale Anforderungen (messbar)

ID	Anforderung	Beschreibung	PURPS-Kategorie
NF-01	Offline-Nutzung	App funktioniert zu 98 % offline ohne Netzwerkzugriff	Portability
NF-02	Energieverbrauch	App verbraucht max. 5 % Akku pro Stunde im Überwachungsmodus	Performance
NF-03	Sicherheit	Keine Speicherung in der Cloud, alle Daten lokal (Fotos, Kontakte)	Security
NF-04	Performance	Sensorreaktion < 300 ms nach Bewegung	Performance
NF-05	Kompatibilität	Android ab Version 10, iOS ab Version 14	Portability
NF-06	Barrierefreiheit	VoiceOver + erhöhter Kontrast unterstützt	Usability
NF-07	Usability	90 % der Benutzer können die App ohne Anleitung bedienen	Usability

Akteur: Benutzer (User) - Die Person, die die App auf ihrem Smartphone installiert hat und zur Überwachung von Bewegungen und zum Senden von Notfallmails verwendet.

3. Anwendungsfalldiagramm

Das Anwendungsfalldiagramm visualisiert die Interaktion zwischen dem **Benutzer** (einziger Akteur) und den Kernfunktionen der App. Es zeigt:

1. **Direkte Benutzerinteraktionen** (Pfeile vom Akteur):
   - Aktivierung der Bewegungserkennung (F-01)
   - Verwaltung von Notfallkontakten (F-02)
   - Nutzung der Pocket-Erkennung (F-06)
   - Manuelle Testfunktionen (F-07)
2. **Automatisierte Abläufe** (interne Pfeile):
   - Bewegungserkennung (F-01) löst den Countdown (F-05) aus
   - Ununterbrochener Countdown (F-05) startet die Dual-Kamera (F-03)
   - Kameraaufnahmen (F-03) initiieren Notfall-E-Mail-Versand (F-04)

Schlüsselaussagen:

- Der Benutzer kontrolliert nur Initialisierungs- und Konfigurationsfunktionen

- Der Kernschutzmechanismus (F-01 → F-05 → F-03 → F-04) läuft automatisch
- Testfunktionen (F-07) stehen isoliert für Diagnosezwecke

4. Testkonzept

4.1 Testumgebung

- **Geräte:** iPhone SE (iOS 15), Samsung Galaxy A52 (Android 12)
- **Emulatoren:** Android Emulator (Pixel 5), Expo Go (iOS/Android)
- **Testtools:** Jest, react-native-testing-library, Expo Test Suite

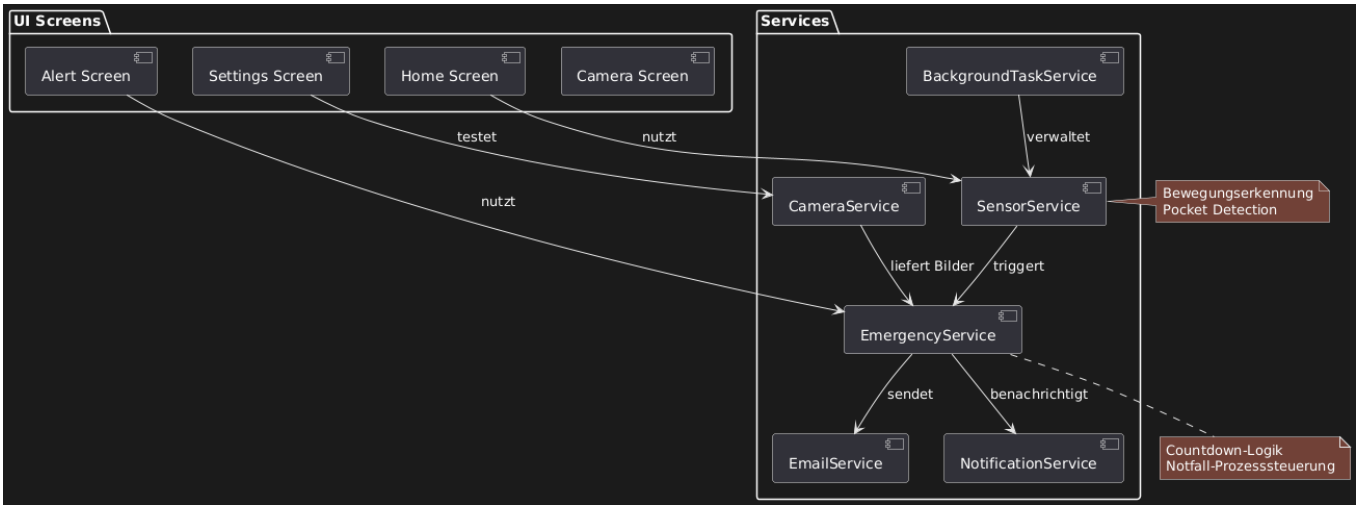
4.2 Testarten

Testart	Beschreibung
Unittests	Einzelne Services (Sensorlogik, Kamera-Service)
Integrationstests	Zusammenspiel Kamera + EmergencyService
Manuelle Tests	Real-World Tests mit echten Geräten
Black-Box	Nutzungsszenarien aus Sicht der Nutzer
White-Box	Tests auf Service-/Modul-Ebene

4.3 Detaillierte Testfälle

ID	Vorbedingungen	Schritte	Erwartetes Ergebnis
TC-01	1. App installiert 2. Bewegungserkennung deaktiviert	1. Home Screen öffnen 2. Bewegungserkennung aktivieren	Statusanzeige wechselt zu "Aktiviert"
TC-02	1. Bewegungserkennung aktiv 2. Kontakte vorhanden	1. Gerät schütteln 2. Countdown beobachten	5s-Countdown erscheint mit Abbruchbutton
TC-03	1. Countdown läuft	1. Countdown abwarten	Fotos beider Kameras im Speicher
TC-04	1. Kontakt vorhanden	1. Einstellungen öffnen 2. Kontakt löschen	Kontakt verschwindet aus Liste
TC-05	1. Kamera-Permission verweigert	1. Kamera-Funktion aufrufen	Fehlermeldung "Kamerazugriff benötigt"

5.1 Komponentendiagramm



Das Komponentendiagramm illustriert die **modulare Architektur** der App mit klarer Aufgabentrennung:

A. UI-Schicht (Obere Ebene)

- **Home Screen**: Zentrale Steuerung der Überwachungsfunktionen
- **Alert Screen**: Verarbeitung des Notfallprozesses (Countdown + Kamera)
- **Settings Screen**: Verwaltung von Kontakten und Sensoreinstellungen
- **Camera Screen**: Manueller Kamerazugriff (Testfunktion)

B. Serviceschicht (Untere Ebene)

Service	Verantwortlichkeit	Abhängigkeiten
`SensorService`	Bewegungserkennung & Pocket-Detection	→ EmergencyService
`CameraService`	Steuerung der Dual-Kamera	→ EmergencyService
`EmergencyService`	Orchestrierung des Notfallprozesses	→ EmailService, NotificationService
`EmailService`	Versand von Notfall-E-Mails	-
`NotificationService`	Lokale Benachrichtigungen	-
`BackgroundTaskService`	Lebenszyklusmanagement	→ SensorService

C. Schlüsselbeziehungen

1. **UI → Services**:
   - Home Screen nutzt SensorService
   - Alert Screen nutzt EmergencyService
   - Settings Screen testet CameraService
2. **Service-Interaktionen**:
   - SensorService *triggert* EmergencyService bei Bewegung
   - CameraService *liefert Bilder* an EmergencyService
   - EmergencyService *nutzt* EmailService für Versand

D. Architekturprinzipien

- **Entkopplung**: UI-Komponenten kennen nur benötigte Services
- **Ereignisgesteuert**: Sensorereignisse starten Notfallkette
- **Wiederverwendbarkeit**: Services sind unabhängig voneinander testbar

5.2 Hauptarchitektur

- **Frontend:** Expo (React Native mit Expo Router)
- **Backend:** Nicht vorhanden, alle Logiken lokal im Gerät integriert
- **Services:** Modulare Dienste (Kamera, Sensor, Mail etc.)

5.3 Expo Screens (Navigation)

- **_layout.tsx:** Initialisierung & SafeAreaProvider
- **index.tsx:** 🏠 Home (Sensorstatus, Start/Stop, Testen)
- **explore.tsx:** ⚙️ Einstellungen (Kontakte, Thresholds)
- **alert.tsx:** 🚨 Notfall-Screen mit Countdown und Kamera
- **camera.tsx:** 📸 Manuelle Fotoaufnahme
- **+not-found.tsx:** Fehlerbildschirm

5.4 Core Services

- **sensorService.ts:** Bewegung & Pocket Detection
- **cameraService.ts:** Dual-Kamera Steuerung
- **emergencyService.ts:** Countdown + E-Mail Logik
- **emailService.ts:** Versand mit Anhängen
- **notificationService.ts:** Lokale Benachrichtigungen
- **backgroundTaskService.ts:** Hintergrund-Erkennung & Wiederaufnahme

5.5 UI-Komponenten

- React Native Paper + eigene Komponenten:
  - `ThemedText`, `ThemedView`
  - `HapticTab`, `IconSymbol`, `ParallaxScrollView`

5.6 Feature-Matrix (Auszug)

Komponente	Features
Home	Bewegungsüberwachung, Auto-Mode, Status, Tests
Alert	Countdown, Dual-Kamera, Notfallversand
Explore	Kontakte, Sensor-Settings, E-Mail-Test
Sensor	Pocket-Detection, Auto-Trigger, Cooldown
Kamera	Dual-View, Timeout, Fallback, Speicherung

5.7 Datenfluss

1. Pocket Detection → Aktivierung
2. Bewegung → Countdown → Kamera → EmergencyService
3. EmergencyService → E-Mail → Kontakte

6. Schichtentrennung

Schicht	Komponenten	Verantwortlichkeit
Präsentation	Screens (index.tsx, alert.tsx...)	UI-Rendering, Nutzerinteraktion
Anwendungslogik	Services (sensorService.ts...)	Geschäftslogik, Datenverarbeitung
Daten	Lokale Speicherung (AsyncStorage)	Persistente Datenspeicherung
Gerätezugriff	Native Module (Expo Sensors, Camera)	Hardware-Kommunikation

Trennungsprinzipien:

1. UI-Komponenten enthalten keine Geschäftslogik
2. Services sind unabhängig von UI-Frameworks
3. Datenmodelle separat definiert (Contact.ts)
4. Hardwarezugriff gekapselt in Services

7. Rahmenbedingungen-Erfüllung

Kriterium	Punkte	Umsetzung in PocketGuardian
Sensor auslesen	5	Beschleunigungssensor für Bewegungserkennung
Hintergrundoperationen	4	BackgroundTaskService für kontinuierliche Überwachung
Systemapplikationen	2	Kamera- und E-Mail-Integration
Persistente Datenablage	2	Lokale Speicherung von Kontakten und Einstellungen
Mehrere Views	1	5 verschiedene Screens
Total	**14**	
