Pflichtenheft: Logistiksystem

Dieses Pflichtenheft beschreibt die technische Umsetzung eines Logistiksystems, bestehend aus einer Fahrer-App, einer Disponenten-Weboberfläche, ERP-Integration, Echtzeit-Tracking und umfassenden Sicherheitsmaßnahmen.

1. Einleitung

Das Logistiksystem soll die Effizienz der Tourenplanung, -durchführung und -überwachung verbessern. Es integriert bestehende ERP-Daten, ermöglicht Echtzeit-Tracking und gewährleistet hohe Datensicherheit.

2. Architektur

2.1. Gesamtarchitektur

- **Microservices-Architektur:** Das System wird in lose gekoppelte, unabhängige Dienste unterteilt. Dies fördert Skalierbarkeit, Wartbarkeit und die einfache Integration neuer Funktionen.
- **API Gateway:** Ein zentraler Einstiegspunkt für alle Client-Anfragen, der Authentifizierung, Routing und Lastverteilung übernimmt.
- **Nachrichten-Broker:** Für asynchrone Kommunikation zwischen Microservices und zur Datenübertragung (z.B. Echtzeit-Updates).
- **Datenbanken:** Jedes Microservice besitzt idealerweise seine eigene Datenbank (Polyglot Persistence), passend zu seinen Anforderungen.

2.2. Komponenten und Dienste

- **User & Authentication Service:** Verwaltet Benutzer (Fahrer, Disponenten), Rollen und Authentifizierung.
- **Tour Management Service:** Verwaltet Touren, Routen, Lieferstatus und Zuweisung von Fahrern.
- **Vehicle Tracking Service:** Empfängt und verarbeitet Echtzeit-Positionsdaten der Fahrzeuge.
- **ERP Integration Service:** Verantwortlich für die Kommunikation mit dem bestehenden ERP-System.
- **Notification Service:** Sendet Push-Benachrichtigungen an Fahrer und aktualisiert die Weboberfläche der Disponenten.
- **Reporting Service:** Generiert Berichte über Touren, Lieferzeiten etc.

3. Technologien

3.1. Plattformen und Frameworks

- **Fahrer-App:**
    - **Plattform:** Android (Native App)
    - **Programmiersprache:** Kotlin
    - **Framework:** Android SDK
- **Disponenten-Weboberfläche:**
    - **Frontend:** React mit Next.js (für SSR und SEO)
    - **Styling:** Tailwind CSS
    - **Kartendarstellung:** Mapbox GL JS oder OpenLayers (für interaktive Karten und Tracking-Visualisierung)
- **Backend (Microservices):**
    - **Programmiersprache:** Node.js (mit Express.js) oder Java (mit Spring Boot) – Wahl abhängig von Team-Expertise und Ökosystem-Präferenzen.
    - **Datenbanken:**
        - PostgreSQL (für relationale Daten wie Benutzer, Touren)
        - Redis (für Caching und schnelle Echtzeit-Daten)
        - MongoDB (optional für flexible Daten wie Audit-Logs)
- **Nachrichten-Broker:** RabbitMQ oder Apache Kafka
- **Echtzeit-Kommunikation:** WebSockets (für Disponenten-Weboberfläche), MQTT (für Fahrer-App und GPS-Geräte)

3.2. Datenübernahme aus dem alten ERP-System

- **Initialmigration:**
    - Einmaliger Datenexport aus dem alten ERP-System (z.B. CSV, XML, JSON).
    - Entwicklung eines dedizierten Import-Tools im `ERP Integration Service`, das die exportierten Daten validiert und in die neuen Datenbanken des Logistiksystems importiert.
- **Laufende Synchronisation:**
    - **Schnittstellen:** Das `ERP Integration Service` stellt eine REST API bereit, über die das ERP-System neue Aufträge, Kunden oder Produktinformationen an das Logistiksystem übermitteln kann.
    - **Polling/Webhook:** Alternativ kann das Logistiksystem das ERP-System in regelmäßigen Abständen nach Änderungen abfragen (Polling) oder das ERP-System sendet bei Änderungen Webhooks an das Logistiksystem.

- **Datenformate:** JSON oder XML für den Datenaustausch.

3.3. Echtzeit-Tracking der Fahrzeuge

- **GPS-Daten-Erfassung:**
  - Fahrer-App sendet regelmäßig (z.B. alle 5-10 Sekunden) GPS-Koordinaten an das Backend.
  - Alternativ: Dedizierte GPS-Tracker im Fahrzeug, die Daten via MQTT an einen MQTT-Broker senden.
- **Datenübertragung:**
  - **MQTT:** Für die Übertragung von GPS-Daten von der Fahrer-App oder externen GPS-Geräten an den `Vehicle Tracking Service`. MQTT ist leichtgewichtig und effizient für IoT-Szenarien.
  - **WebSockets:** Der `Vehicle Tracking Service` stellt eine WebSocket-Schnittstelle bereit, über die die Disponenten-Weboberfläche Echtzeit-Positionsdaten abonnieren kann.
- **Visualisierung:** Die Disponenten-Weboberfläche nutzt eine Kartenbibliothek (Mapbox GL JS/OpenLayers), um die aktuellen Positionen der Fahrzeuge auf einer Karte darzustellen und deren Bewegung in Echtzeit zu verfolgen.

4. Funktionen und Module

4.1. Fahrer-App (Android)

- **Login:** Sichere Authentifizierung über den `User & Authentication Service`.
- **Touren-Abruf:** Anzeige der zugewiesenen Touren für den aktuellen Tag/Woche.
- **Touren-Details:** Anzeige von Lieferadressen, Kontaktinformationen, Paketdetails.
- **Lieferstatus-Aktualisierung:** Funktionen zum Setzen des Status (z.B. "Unterwegs", "Angekommen", "Zugestellt", "Problem").
- **GPS-Tracking:** Kontinuierliches Senden der Standortdaten im Hintergrund.
- **Offline-Fähigkeit:** Zwischenspeicherung von Toureninformationen und Status-Updates für den Fall von Netzwerkausfällen.

4.2. Disponenten-Weboberfläche

- **Login:** Sichere Authentifizierung über den `User & Authentication Service`.
- **Tourenplanung:**
  - Erstellung neuer Touren mit Start- und Zielpunkten, Zwischenstopps.
  - Zuweisung von Fahrern und Fahrzeugen zu Touren.
  - Drag-and-Drop-Funktionalität auf der Karte zur Routenoptimierung.

- **Tourenübersicht:** Dashboard mit allen aktiven und geplanten Touren.
- **Echtzeit-Tracking-Karte:** Interaktive Karte zur Visualisierung der Fahrzeugpositionen und Routen.
- **Statusübersicht:** Anzeige des aktuellen Lieferstatus aller Touren.
- **Berichte:** Zugriff auf Leistungsberichte (z.B. Lieferzeiten, gefahrene Kilometer).

5. Sicherheit und Zugriffskontrolle

5.1. Authentifizierung

- **Standard:** OAuth 2.0 und OpenID Connect (OIDC) für die Authentifizierung von Fahrern und Disponenten.
- **Implementierung:** Ein Identity Provider (z.B. Keycloak, Auth0 oder ein selbst entwickelter `User & Authentication Service`) verwaltet die Benutzeridentitäten.
- **Token-basierte Authentifizierung:** JWTs für den Zugriff auf die Microservices.

5.2. Autorisierung (Rollenrechte)

- **Rollenbasierte Zugriffskontrolle (RBAC):**
    - **Rolle `Fahrer`:** Darf nur eigene Touren abrufen und deren Status aktualisieren, GPS-Daten senden.
    - **Rolle `Disponent`:** Darf Touren planen, Fahrern zuweisen, alle Touren und Fahrzeuge verfolgen, Berichte einsehen.
    - **Rolle `Admin`:** Volle Systemzugriffsrechte (Benutzerverwaltung, Konfiguration).
- **Implementierung:** Jede API-Anfrage wird vom API Gateway und/oder den einzelnen Microservices auf gültige Rollen und Berechtigungen geprüft.

5.3. Datensicherheit

- **Verschlüsselung in Transit:** Alle Kommunikationen (zwischen Clients und API Gateway, zwischen Microservices) erfolgen über TLS/SSL (HTTPS, WSS, MQTTS).
- **Verschlüsselung at Rest:** Sensible Daten in den Datenbanken werden verschlüsselt gespeichert (Transparent Data Encryption oder anwendungsspezifische Verschlüsselung).
- **Eingabevalidierung:** Strikte serverseitige Validierung aller Benutzereingaben, um Injections und andere Angriffe zu verhindern.
- **Logging und Monitoring:** Umfassendes Logging von Zugriffsversuchen und Systemereignissen zur Erkennung von Anomalien.
- **Backup und Wiederherstellung:** Regelmäßige Backups der Datenbanken und Konfigurationsdateien mit einem getesteten Wiederherstellungsplan.

6. Teststrategie

6.1. Teststufen

- **Unit Tests:** Für einzelne Funktionen und Logik der Microservices, Frontend-Komponenten und App-Module.
- **Integration Tests:** Überprüfung der Interaktion zwischen Microservices, Datenbanken und externen APIs (ERP, Zahlungs-Gateways).
- **End-to-End Tests:** Automatisierte Tests, die den gesamten Workflow aus Sicht des Benutzers simulieren (z.B. Tourenplanung durch Disponent, Abruf und Status-Update durch Fahrer).
- **Performance Tests:** Lasttests zur Überprüfung der Systemstabilität und Skalierbarkeit unter hoher Last (z.B. viele gleichzeitige GPS-Updates).
- **Sicherheitstests:** Penetrationstests, Schwachstellen-Scans, Code-Audits.

6.2. Testumgebungen

- **Entwicklungsumgebung:** Lokale Tests während der Entwicklung.
- **Testumgebung (Staging):** Eine Umgebung, die der Produktionsumgebung ähnelt, für Integrationstests und manuelle Abnahmen.
- **Produktionsumgebung:** Kontinuierliches Monitoring und A/B-Tests (falls zutreffend).

6.3. Testautomatisierung

- Einsatz von CI/CD-Pipelines (Continuous Integration/Continuous Deployment), die automatisierte Tests bei jeder Code-Änderung ausführen.