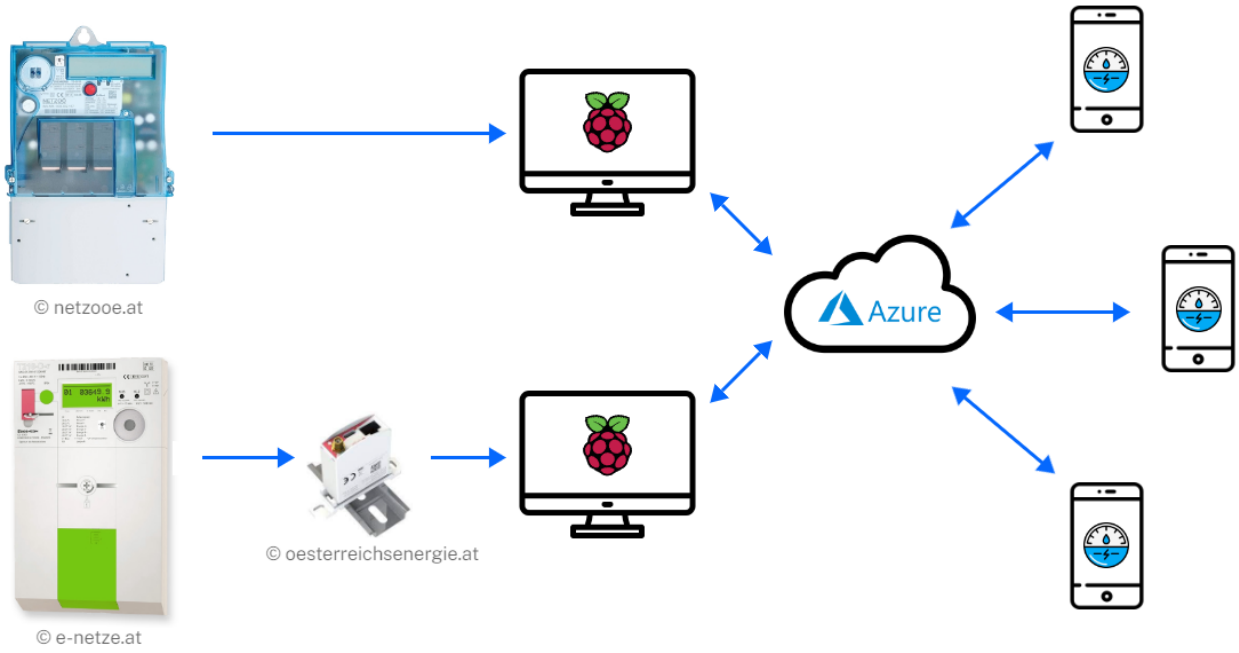
Architektur

Allgemein

- Smart Meter
  - digitales Messgerät für den Energieverbrauch/-erzeugung
  - Schnittstelle zwischen Haushalt und Netzbetreiber
    - weiters auch zwischen Haushalt und unserer Plattform
  - Kundenschnittstelle **AMIS Zähler** der Energie AG
    - Infrarotschnittstelle
    - liefert jede Sekunde einen Messwert
    - Verschlüsselung der Daten mittels AES-Key
    - künftig einheitliche Schnittstelle in Österreich
- Raspberry
  - liest die Energiedaten mittels der Kundenschnittstelle vom Smart Meter aus
    - C++ Service (programmiert von Energie AG und abgeändert durch uns)
      - persistiert historische Daten alle 15 min
      - speichert Sekundendaten für die letzten 90 Sekunden
  - .NET Core Backend
    - bearbeitet die Energiedaten entsprechend für die Plattform
    - Datenaustausch von/zur Cloud mittels SignalR
      - vor allem Echtzeitdaten
    - Datenaustausch zur Cloud teilweise mit REST (ohne Echtzeitrelevanz)
  - SQLite3 Datenbank
  - GoQuorum Blockchain Node
    - Jeder Raspberry befindet sich in einem privaten Ethereum Netzwerk
    - Dient zur dezentralen Speicherung von Einwilligungserklärungen
- Cloud (Azure)
  - .NET Core Backend
    - zentrale Logik und Zugangspunkt
    - Handler für alle Energiegemeinschaften und deren Mitglieder inklusive Smart Meter
  - MS-SQL Datenbank
- Endgeräte (Android)
  - Schnittstelle zum Mitglied
  - Echtzeitdaten werden mittels SignalR empfangen
  - andere Daten werden über REST-Requests abgefragt
  - erhalten Push-Nachrichten mittels Firebase Cloud Messaging (FCM)

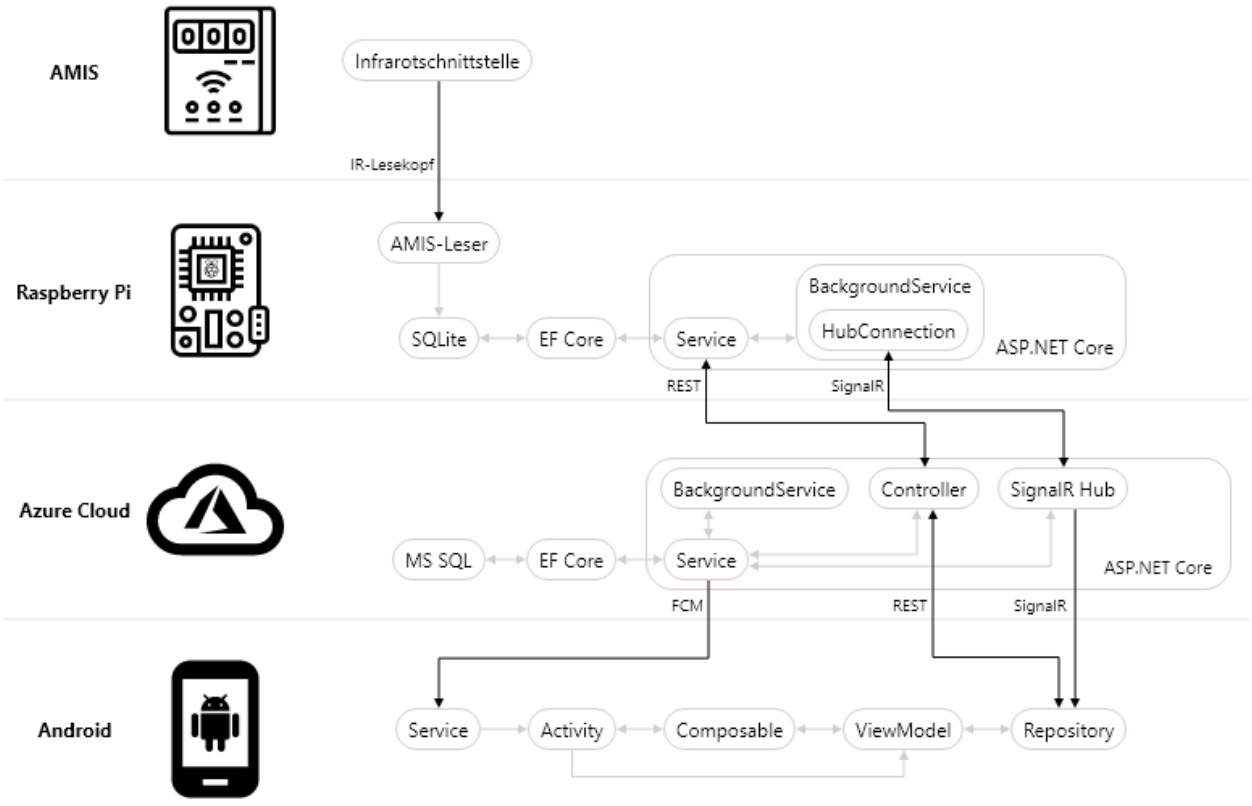


© netzooe.at

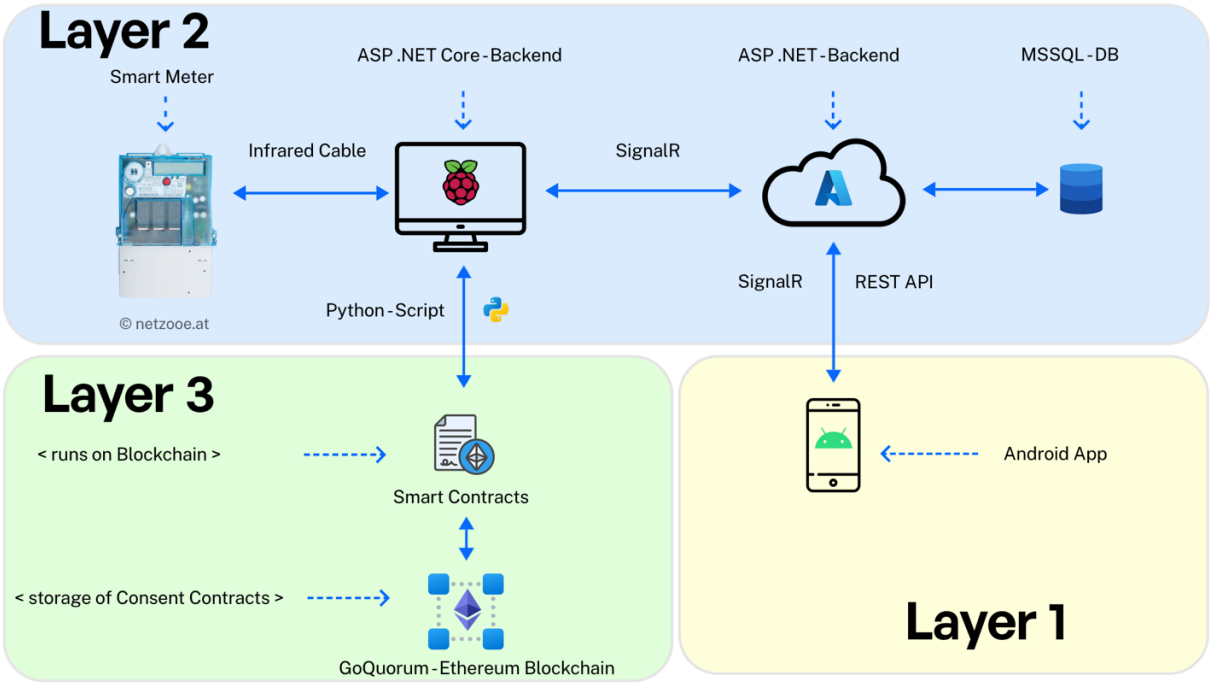
© oesterreichsenergie.at

© e-netze.at

Optimierung



Consent Management

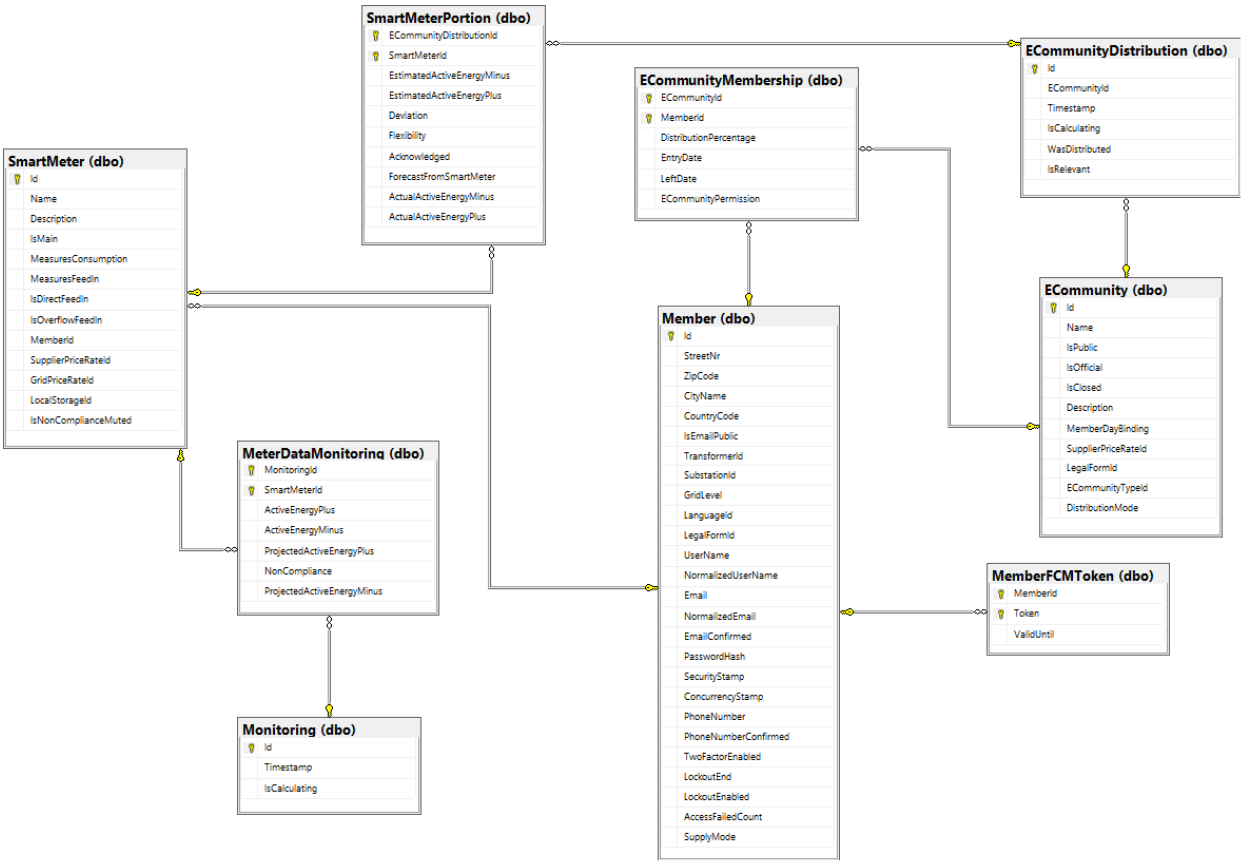


Datenmodelle

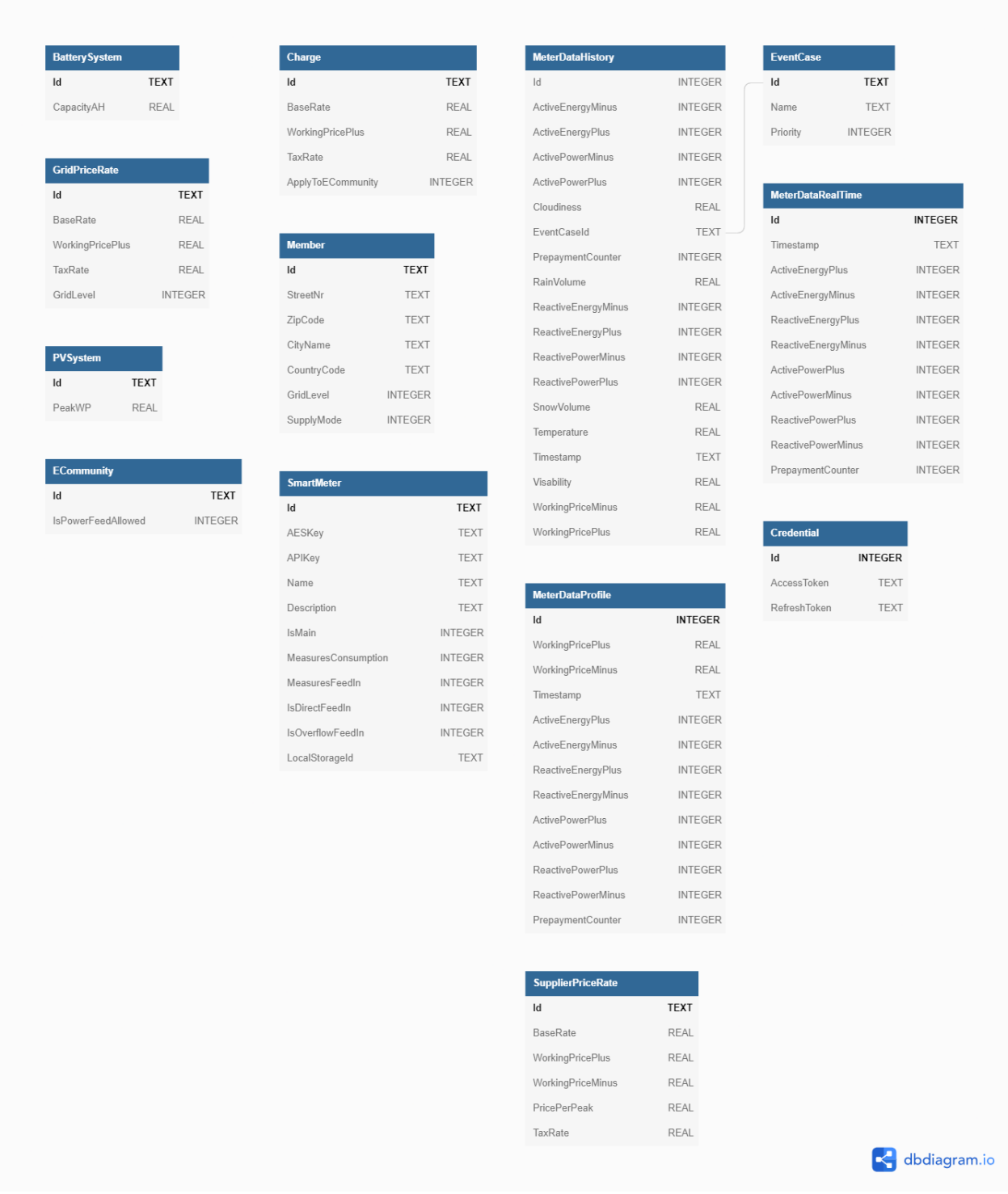
Beschreibung der Attribute kann man sich im Source Code ansehen.

Cloud (Azure)

Anm.: Nicht verwendete Tabellen sind in der folgenden Abbildung nicht dargestellt, können aber im Quellcode nachgeschlagen werden.



Lokal (Raspberry Pi)



App (Android)

```kotlin
@Entity(tableName = "tile")
data class Tile(
    @PrimaryKey
    var tileId: String,
    var region: String,
    var action: String,
    var value: String,
    var isVisible: Boolean,
    var isLargeTile: Boolean = false,
    var order: Int,
    var colorId: Int) {}
```

```kotlin
@Entity(tableName = "smart_meter")
data class SmartMeter(
    @PrimaryKey var ID: String,
    var name: String,
    var description: String
)
```

```kotlin
@Entity (tableName = "member")
data class Member(
    @PrimaryKey var memberId: String,
    var username: String,
    var email: String,
    var password: String,
    var accessToken: String,
    var refreshToken: String,
    var languageName: String)
```

```kotlin
@Entity(tableName = "blockchain_balance")
data class BlockchainBalance(

    @PrimaryKey(autoGenerate = true)
    val id: Int? = null,
    val received: String? = null,
    val sent: String? = null,
    var balance: String? = null
)
```

```kotlin
@Entity(tableName = "contract")
data class ConsentContract (
    @PrimaryKey
    @SerializedName("ContractId")
    var contractId: kotlin.String,
    @SerializedName("State")
    var state: kotlin.Int? = null,
    @SerializedName("AddressContract")
    val addressContract: kotlin.String? = null,
    @SerializedName("AddressProposer")
    val addressProposer: kotlin.String? = null,
    @SerializedName("AddressConsenter")
    val addressConsenter: kotlin.String? = null,
    @SerializedName("StartEnergyData")
    val startEnergyData: kotlin.String? = null,
    @SerializedName("EndEnergyData")
    val endEnergyData: kotlin.String? = null,
    @SerializedName("ValidityOfContract")
    val validityOfContract: kotlin.String? = null,
    @SerializedName("PricePerHour")
    val pricePerHour: kotlin.String? = null,
    @SerializedName("TotalPrice")
    val totalPrice: kotlin.String? = null,
    @SerializedName("DataUsage")
    val dataUsage: kotlin.Int? = null,
    @SerializedName("TimeResolution")
    val timeResolution: kotlin.Int? = null
)
```

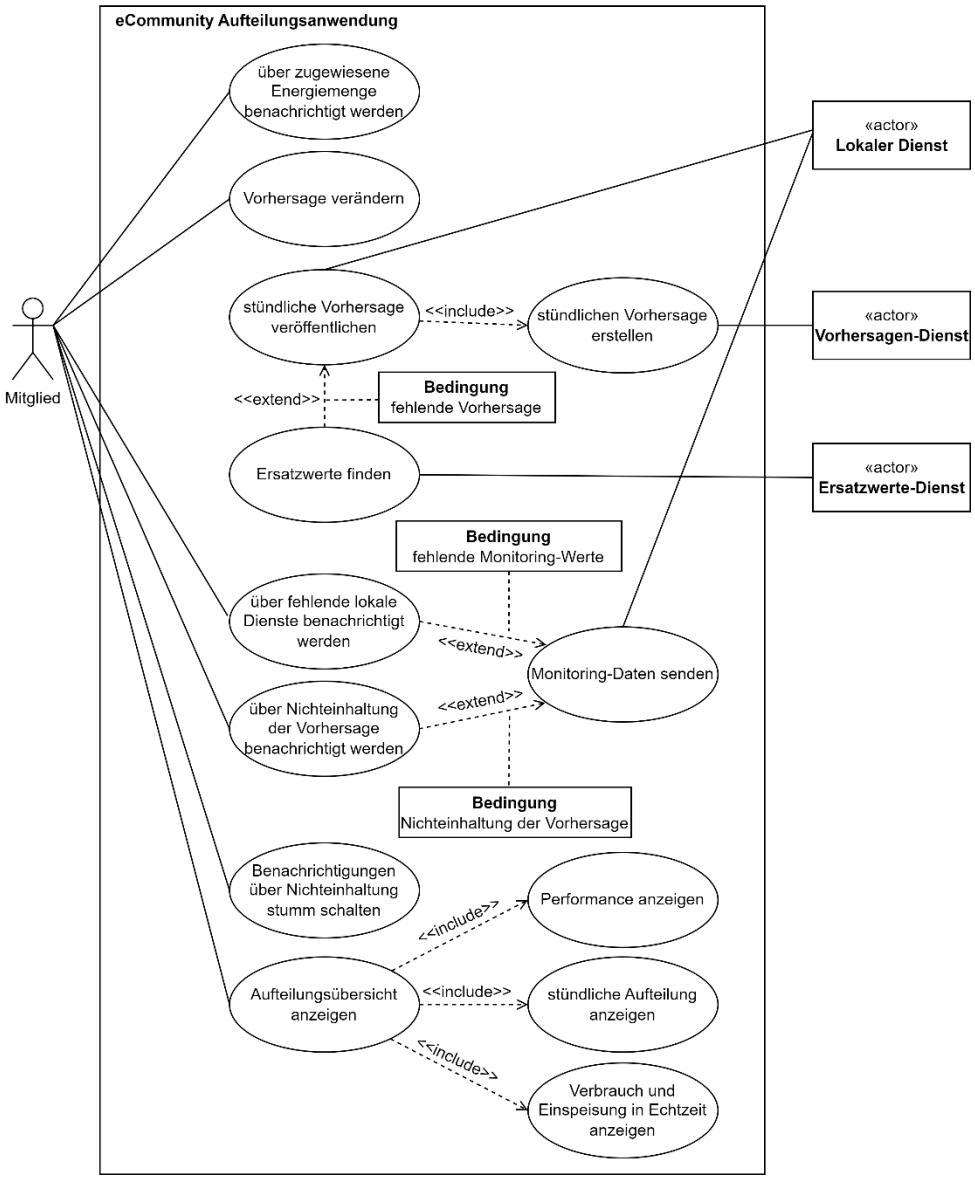
```kotlin
@Entity(tableName = "meter_data_hist_contract")
data class MeterDataHistContract(

    @PrimaryKey(autoGenerate = true)
    @SerializedName("EnergyId")
    val id: Int? = null,
    var contractId: String? = null,
    @SerializedName("id")
    val dataId: Int? = null,
    @SerializedName("timestamp")
    val timeStamp: String? = null,
    val activeEnergyPlus: Int? = null,
    val activeEnergyMinus: Int? = null,
    val reactiveEnergyPlus: Int? = null,
    val reactiveEnergyMinus: Int? = null,
    val activePowerPlus: Int? = null,
    val activePowerMinus: Int? = null,
    val reactivePowerPlus: Int? = null,
    val reactivePowerMinus: Int? = null,
    val prepaymentCounter: Int? = null
)
```

Use Cases

Optimierung

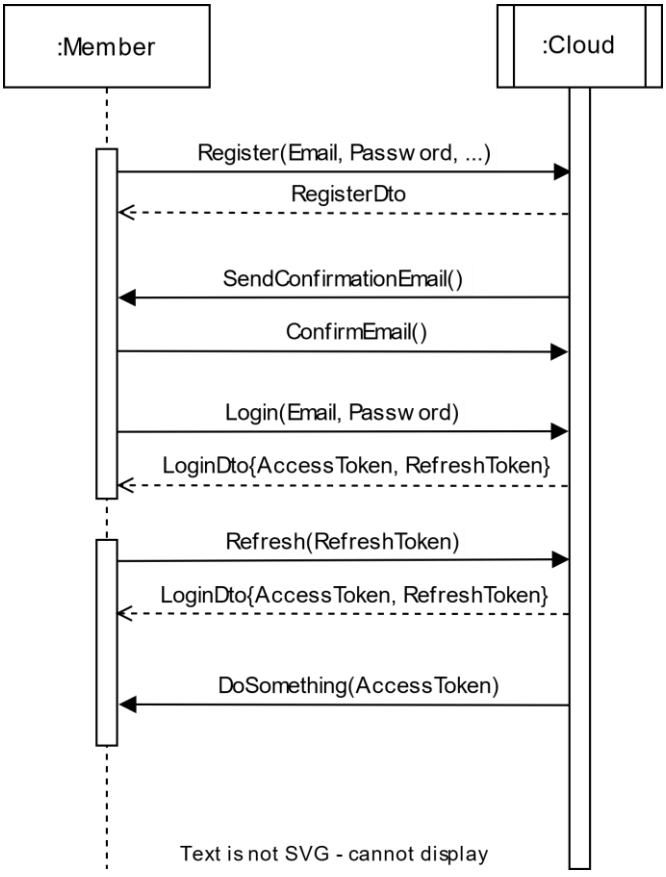
➔ Beschreibung siehe Bachelorarbeit Fischer



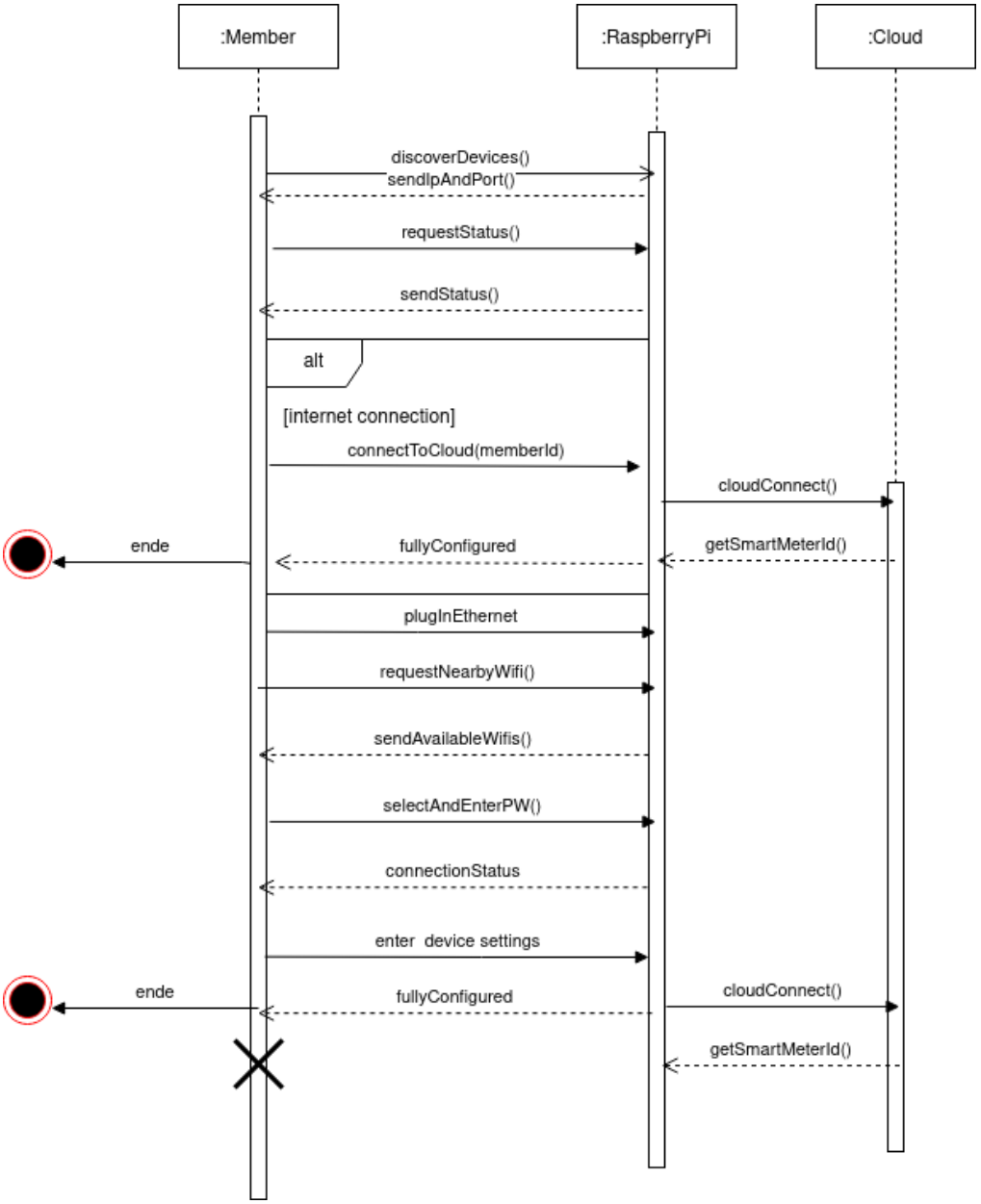
Abläufe

Authentifizierung

- Login
  - Member muss E-Mail bestätigt haben
  - E-Mail und Passwort müssen natürlich übereinstimmen
  - bei 3 fehlgeschlagenen Versuchen
    - Member für 5 Minuten gesperrt
- Access Token
  - gültig für 10 Minuten
  - Zugriff auf Ressourcen (REST und SignalR)
- Refresh Token
  - gültig für 90 Tage
  - man erhält neuen Access und Refresh Token



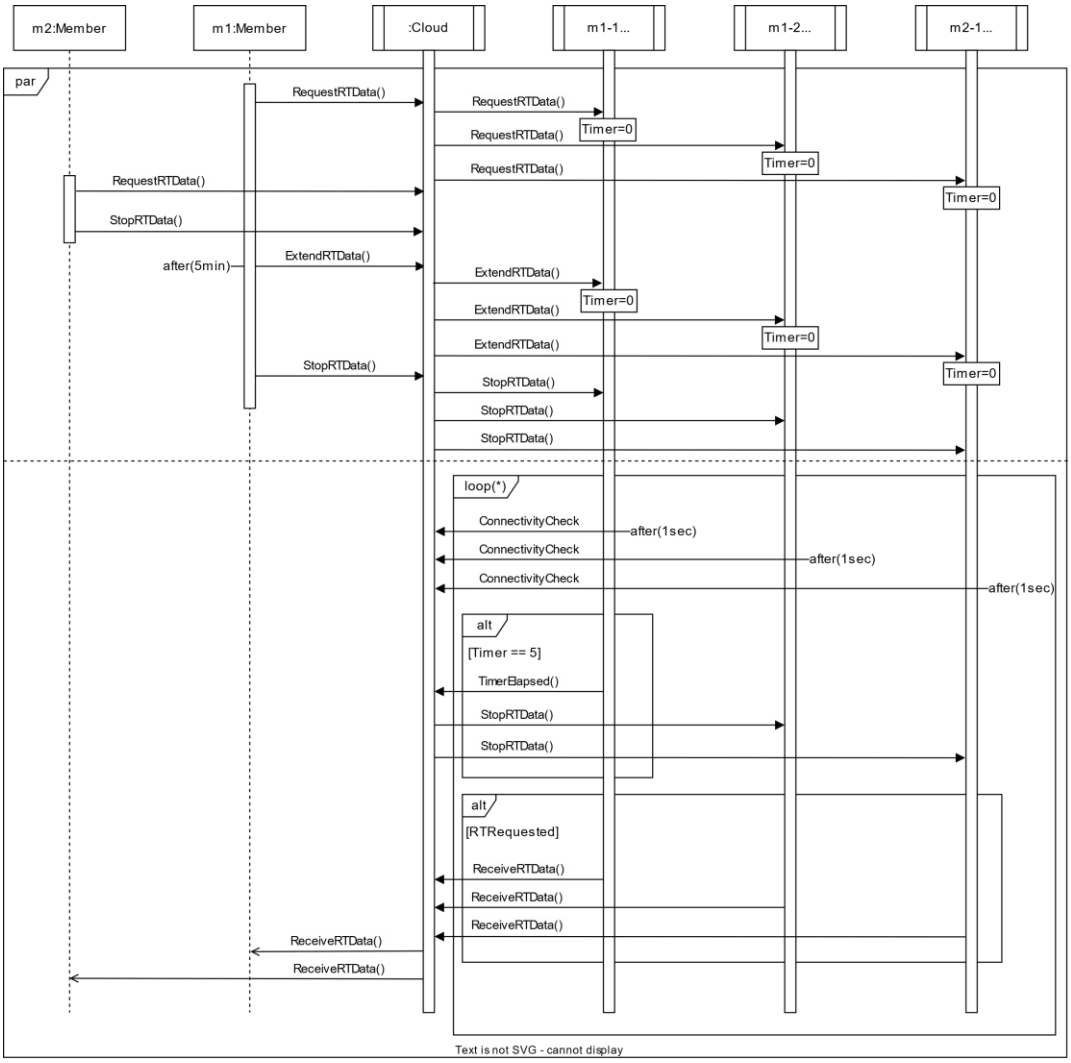
Pairing



Real Time

- Abwicklung des Echtzeit-Traffics mittels SignalR
- `RequestRTData`
  - Endgerät-Session starten
  - Raspberry-Sessions starten
    - falls nicht bereits gestartet
    - bei ersten Gerät von Energiegemeinschaft
- `ExtendRTData`
  - Endgeräte müssen alle 5 Minuten ihre Session verlängern

        - Verhinderung von unnötigen Traffic
- `StopRTData`
    - Endgerät-Session stoppen
    - Raspberry-Sessions stoppen
        - falls alle Endgerät-Sessions geschlossen
- jede Sekunde durchläuft ein Raspberry die Schleife
- `ConnectivityCheck`
    - überprüft ob SignalR-Verbindung noch offen ist
    - falls geschlossen: neu verbinden
- `TimerElapsed`
    - Timer ist abelaufen
    - Echtzeit-Session stoppen
    - `ExtendRTData` wurde nicht aufgerufen
- `ReceiveRTData`
    - aktueller Messwert an Cloud senden



Optimierung

- Stündliche Vorhersage
    - 10 Minuten vor vollen Stunde (e.g. 11:50 Uhr)
    - Vorhersage wird vom Server angefragt und anschließend vom lokalen Dienst gesendet
- Neue Energiezuteilung
    - 5 Minuten vor vollen Stunde (e.g. 11:55 Uhr)
    - wird an Mitglied gesendet
        - dieser kann ggf. die Flexibilität verändern
        - nach Veränderung der Flexibilität wird die Nachricht erneut versendet
- Finale Energiezuteilung
    - volle Stunde (e.g. 12:00 Uhr)
    - wird an Mitglied gesendet
- Monitoring
    - alle vollen 5 Minuten
    - Monitoring-Daten werden vom Server angefragt und anschließend vom lokalen Dienst gesendet
        - Überprüfung auf Nichteinhaltung und auf fehlende Einträge (offline)

➔ detaillierte Beschreibung siehe Bachelorarbeit Fischer