# Distanzberechnungsformell

Quelle: Shah, R. (2021). *Convert RSSI Value of the BLE (Bluetooth Low Energy) Beacons to Meters.* <https://medium.com/beingcoders/convert-rssi-value-of-the-ble-bluetooth-low-energy-beacons-to-meters-63259f307283>

# Trilateration

Mit Hilfe der Trilateration lässt sich im Gegensatz zum Näherungsverfahren die genaue Position einer Person in einem Raum bestimmen. Die Trilateration ist ein mathematisches Verfahren zur Bestimmung der Position auf der Grundlage der Abstände zu drei Bezugspunkten. Dieses Verfahren wird auch bei satellitengestützten Systemen zur Positionsbestimmung verwendet. Abbildung 1 demonstriert die Positionsbestimmung mit dem Trilaterationsverfahren. Im zweidimensionalen Raum legt die Entfernung zu einem Beacon einen Kreis fest, in dem sich eine Person befindet. Der Abstand zum zweiten Bezugspunkt legt einen zweiten Kreis fest, dessen Schnittpunkt mit dem ersten Kreis die mögliche Position der Person eingrenzt. Der Abstand zum dritten Bezugspunkt ermöglicht die genaue Berechnung der Position. Die Trilateration kann auch im dreidimensionalen Raum durchgeführt werden, wobei dann anstelle von Kreisen von Kugeln gesprochen wird. Dieses Verfahren kann auch mit mehr als drei Bezugspunkten durchgeführt werden, dann spricht man von Multilateration.

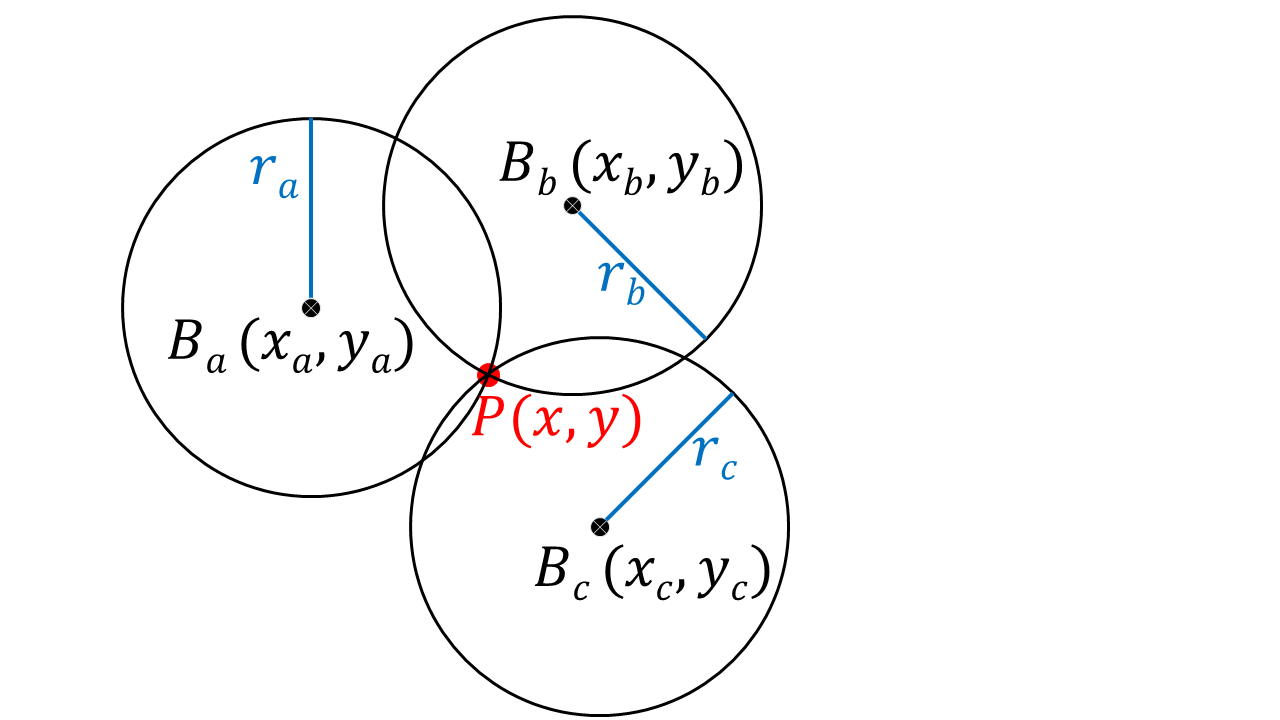


Abbildung 1: Einfache Trilateration

Das Trilaterationsverfahren im zweidimensionalen Raum bedeutet aus mathematischer Sicht einen Punkt zu finden, der drei Kreisgleichungen erfüllt, was der Lösung der in Formel 1 dargestellten drei Gleichungen mit den zwei Unbekannten x und y entspricht.

Formel 1: Trilaterationsgleichungen (Bembenik & Falcman, 2020, S. 56)

Die Unbekannten x und y sind die gesuchten Koordinaten der Position. Die xa, ya, xb, yb, xc und yc stellen die bekannten Koordinaten der drei Referenzpunkte Ba, Bbund Bc dar. Die ra, rb und rc sind die gemessenen Abstände zu den Bezugspunkten. Die Bezugspunkte werden in der Regel in einem lokalen Koordinatensystem der Gebäudeetage oder globalen Koordinatensystem definiert und die Entfernungen in gebräuchlichen Einheiten gemessen.

Um die Trilateration-Methode anwenden zu können, müssen somit die exakten Koordinaten der Beacons bekannt sein. Hierfür kann eine Indoor-Karte erstellt werden, auf welche die installierten Beacons gemappt werden. Eine direkte Entfernungsmessung ist nicht möglich, daher werden andere Merkmale, wie die Ausbreitung von Hochfrequenzsignalen, gemessen und auf deren Grundlage dann die Entfernung berechnet. Die jeweils drei nächsten Beacon-Signale werden zur Berechnung der Position verwendet. Um die Genauigkeit der Lokalisierung zu maximieren, sollten die drei Referenzpunkte, die zur Positionierung verwendet werden, ein gleichseitiges Dreieck bilden. In grossen Beacon-Netzwerken sollten die Referenzknoten symmetrisch und so platziert werden, dass die Position eines neuen Referenzknotens auf der Grundlage der Position der drei ursprünglichen Knoten berechnet werden kann. Dabei kann ein Netzraster mit Beacons an der Decke eingerichtet werden.

Je mehr Beacons verfügbar sind, desto genauer kann die Position geschätzt werden, da die Distanzen zu den jeweils nächsten Beacons kürzer und die Signale weniger anfällig auf Störvariablen sind. Somit kann der Standort der Person über den ganzen Tag hinweg verfolgt und mit den festgelegten Standorten der Patienten können die Behandlungszeiten berechnet werden. Die Trilateration-Methode ist deutlich aufwendiger zu implementieren als der Näherungsansatz, die Positionsermittlung ist jedoch genauer und es wird ermöglicht, die Bewegungsroute einer Person über einen Tag hinweg zu bestimmen.

Bei perfekten Informationen, sprich bei exakt berechneten Distanzen zu Referenzpunkten, liefert Trilateration eine exakte Position, das heisst, dass es einen einzigen Punkt im Schnittpunkt der Kreise in Abbildung 1 gibt. Es ist jedoch wichtig zu beachten, dass es unter realen Bedingungen eher unwahrscheinlich ist, dass sich alle Kreise in genau einem Punkt schneiden. Da die Signale durch verschiedene Störvariablen abgeschwächt werden können, ist es sehr wahrscheinlich, dass der Schnittpunkt der Kreise wie in Abbildung 2 einer Fläche entspricht. Im Fall von einer Verzögerung bei der Verarbeitung der Signale oder falscher Berechnung der Distanzen ist es möglich, dass die drei Kreise wie in Abbildung 3: Trilateration mit zu kurzen Distanzen keinen gemeinsamen Schnittpunkt haben. In diesen Fällen kann die Lösung nicht direkt berechnet werden, da der Satz von Trilaterationsgleichungen keine eindeutige Lösung ergibt und Näherungsverfahren angewendet werden müssen.

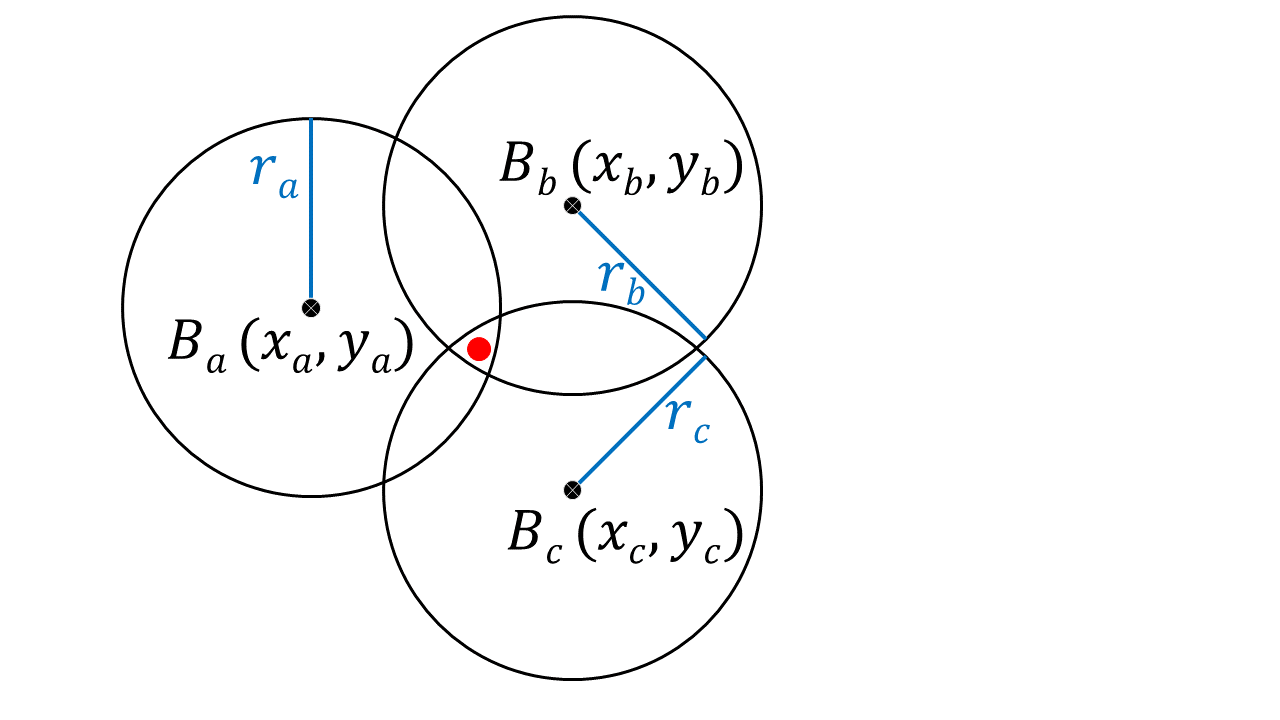


Abbildung 2: Trilateration mit zu langen Distanzen

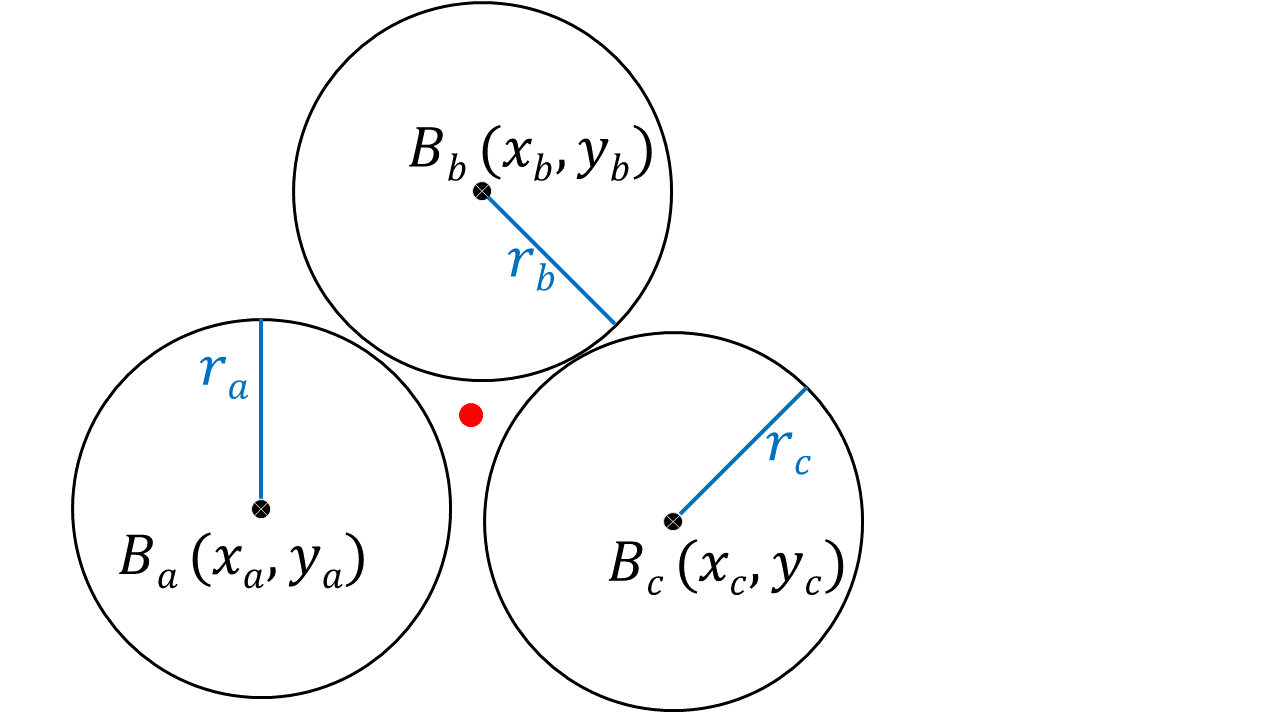


Abbildung : Trilateration mit zu kurzen Distanzen

# Android-Client

Die Benutzeroberfläche der Android Applikation wurde gänzlich neugestaltet, um neue Funktionalitäten zu ermöglichen. Die Abbildung 4 zeigt die neue, aus acht Komponenten bestehende Benutzeroberfläche. Die erste Komponente ist ein Textfeld und zeigt den allgemeinen Applikationsstatus. Die zweite Komponente ist ein Button, welcher den Mock-Login ausführt. Ein Mock-Login autorisiert die ausgehenden REST-Anfragen für 15 Minuten, danach muss der Login erneut durchgeführt werden. Um den Login-Vorgang zu wiederholen, musste die die Applikation zuvor neugestartet werden. Normalerweise würde man bei jeder autorisierten HTTP-Anfrage einen neuen JWT-Token generieren, welcher für weitere 15 Minuten gültig ist. Aufgrund beschränkter Zeit, die für die Entwicklung zur Verfügung stand, und anderer Prioritäten wurde diese Funktionalität jedoch nicht implementiert.

Zuvor war das Scann-Intervall, in welchen nach Signalen gesucht wird, festprogrammiert und konnte nur im Programmcode geändert werden. Um die Auswirkung verschiedener Scann-Intervalle auf die Positionsbestimmung zu untersuchen wurde ein Button (Komponente 4) hinzugefügt, damit das Scann-Intervall zur Applikationslaufzeit geändert werden kann. Beim Betätigen dieses Buttons kann das Scann-Intervall um 100 Millisekunden erhöht werden, wobei der Anfangswert 100 und der maximale wert 1000 Millisekunden sind. Sobald der Maximalwert erreicht ist, wird der Wert beim erneuten Betätigen des Buttons wieder auf den Anfangswert gesetzt. Die dritte Komponente ist ein Textfeld, welches das aktuelle Scann-Intervall in Millisekunden anzeigt. Die fünfte Komponente ist ein Button, welcher das Starten und Beenden des Scannens ermöglicht. In der sechsten Komponente wird die Gesamtanzahl der seit dem Applikationsstart empfangenen Signale angezeigt. In der achten Komponente werden allfällige Errors angezeigt und können mit einem weiteren Button (Komponente 7) wieder gelöscht werden. Bei der Gestaltung der neuen Benutzeroberfläche wurde nur auf die Zweckmässigkeit geachtet.

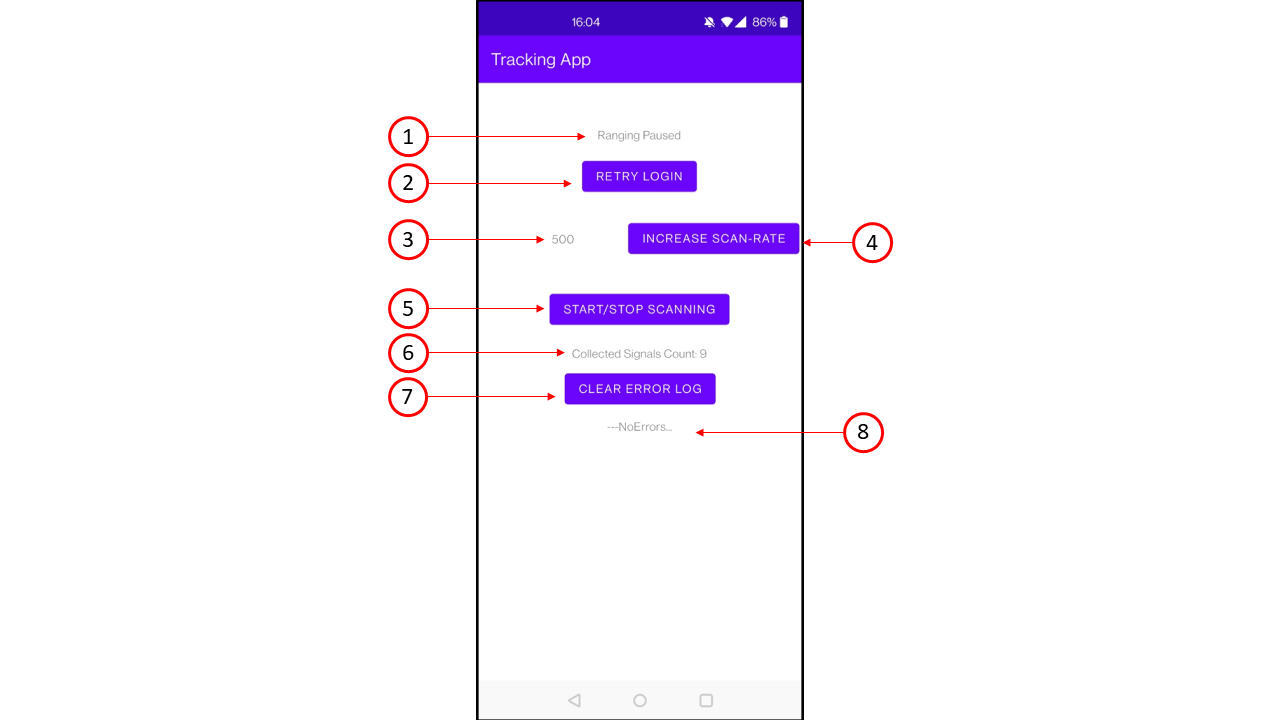


Abbildung 4: Benutzeroberfläche des Android-Clients

Eine weitere Verbesserung der Applikation ist die Integration von Sentry. Sentry ist eine Software, welche unter anderem das Überwachen der Logs, der Errors und der Performance von mobilen Applikationen ermöglicht. Die Daten können dabei mittels der Sentry-Library an den Sentry-Server übermittelt werden und über ein Webbasierte Benutzeroberfläche analysiert werden.

# Server

Der Server wurde grösstenteils überarbeitet. Die Datenbank wird weiterhin beim Starten der Applikation automatisch aus Liquibase-Changesets generiert. Zusätzlich werden danach die Daten der SBB Beacons aus einer CSV-Datei gelesen und in die Datenbank persistiert. In diesem Kapitel wird zuerst die neue Entitätsstruktur dargestellt. Danach werden Implementierte REST-APIs und die Routenberechnung dokumentiert. Zuletzt wird ein Überblick über die Architektur und konfigurierbare Parameter gegeben.

## Entitäten

Das Klassendiagramm in Abbildung 5 stellt die implementierte Entitätsstruktur dar. Ein Admin-Benutzer wird weiterhin für die Autorisierung der REST-Anfragen benötigt. Die Entität Beacon wurde um weitere Attribute erweitert, damit die Daten der SBB-Beacons repräsentiert und gespeichert werden können. Neu implementierte Entitäten sind Device, Signal, Position und Route. Ein Device stellt jeweils einen Mobile-Client dar und ermöglich die Datenerhebung und Positionsberechnung mit mehreren Geräten. Ein Signal beinhaltet alle relevanten Informationen eines Beacon-Signals, wie zum Beispiel den RSSI, Sendeleistung und die Empfangszeit. Für ein Device können, basierend auf den empfangenen Signalen, mehrere Routen berechnet werden, wobei eine Route aus mindestens einer Position bestehen muss und für die Berechnung einer Position drei Beacon-Signale benötigt werden. Für die in Java implementierte Entitätsstruktur wurde eine entsprechende Datenbankstruktur mittels Liquibase-ChangeSets generiert.

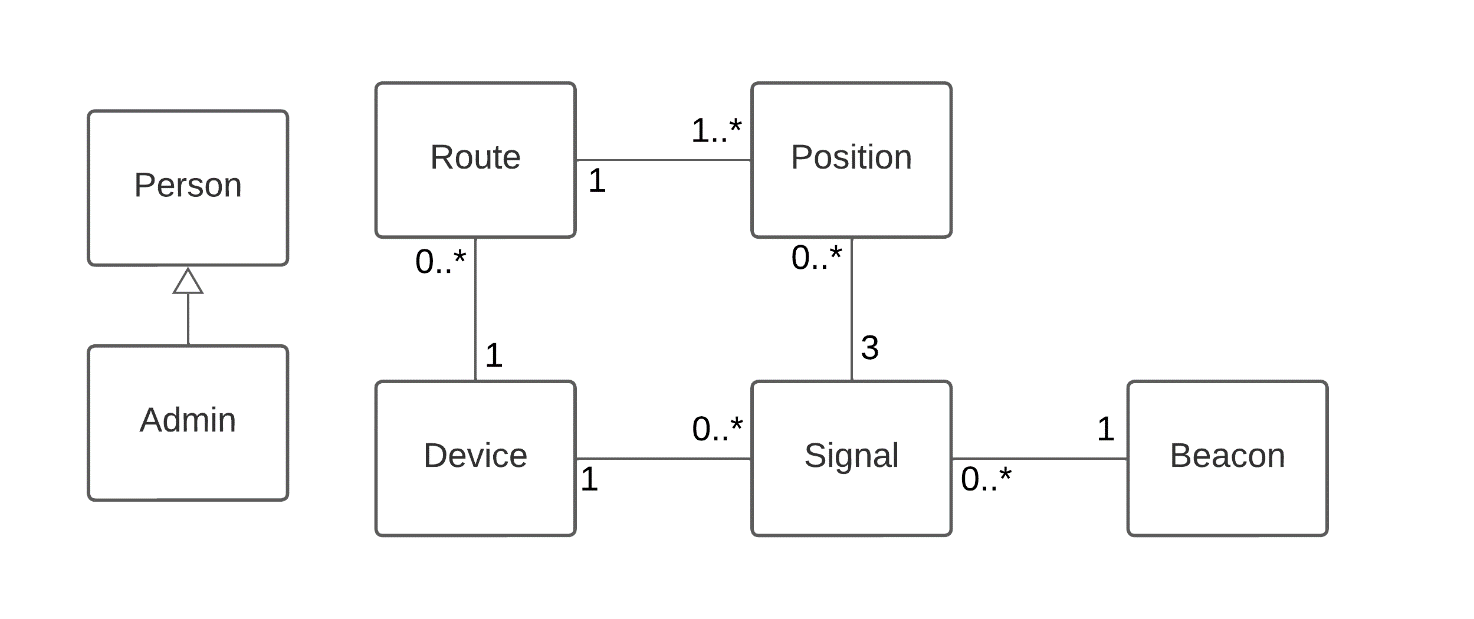


Abbildung 5: Klassendiagramm der neuen Entitätsstruktur

## REST-APIs

In diesem Kapitel werden die implementierten REST-APIs dokumentiert. Alle über die REST-APIs generierten Output-Dateien werden im out-Ordner gespeichert.

### Admin

Der Funktionsumfang des Admin-Rest-Controller wurde nicht verändert. Dieser Controller stellt weiterhin zwei REST-APIs zur Verfügung, welche in Tabelle 1 und Tabelle 2 definiert werden. Hierbei handelt es sich um öffentliche Schnittstellen.

|  |  |
| --- | --- |
|  | Neuen Admin-Benutzer erstellen |
| URI | beacons/api/public/admins/signup |
| HTTP-Methode | POST |
| Body-Parameter | JSON-Objekt, z.B.:  {  "firstName" : "Vorname",  "surname" : "Nachname",  "email" : "email@email.com",  "password" : "123456asd$$"  } |

Tabelle 1: Admins-API: Neuen Admin-Benutzer erstellen

|  |  |
| --- | --- |
|  | Login |
| URI | beacons/api/public/admins/login |
| HTTP-Methode | POST |
| Body-Parameter | JSON-Objekt, z.B.:  {  "email" : "email@email.com",  "password" : "123456asd$$"  } |

Tabelle 2: Admins-API: Login

### Beacon

Zusätzlich zur bereits existierenden Schnittstelle zum Anfordern der Daten aller Beacons wurde der Beacon-Rest-Controller um eine weitere Schnittstelle erweitert, welche alle Beacons mit mindestens einem Signal zurückgibt. Beim Aufruf dieser API werden die Daten dieser Beacons automatisch in vier CSV-Dateien exportiert. Jede Datei beinhaltet Beacons für jeweils eines der Stockwerke des Bahnhofs in Zürich. Die Output-Dateien sind in einem mit Opendatasoft kompatiblen Format strukturiert und ermöglichen die Darstellung der Beacons als Geopunkte.

|  |  |
| --- | --- |
|  | Alle Beacons lesen |
| URI | beacons/api/internal/beacons |
| HTTP-Methode | GET |

Tabelle 3: Beacons-API: Alle Beacons lesen

|  |  |
| --- | --- |
|  | Alle Beacons mit Signalen lesen |
| URI | beacons/api/internal/beacons/matched |
| HTTP-Methode | GET |
| URI-Parameter | String: Output-Dateiname |

Tabelle 4: Beacons-API: Alle Beacons mit Signalen lesen

### Device

Mit dem Device-Rest-Controller wurden zwei neue Schnittstellen implementiert. Die erste Schnittstelle dient zur Registrierung von neuen Devices und wird vom Client immer nach dem Login aufgerufen, damit die erhobenen Signale einem Device zugewiesen werden können. Die zweite Schnittstelle dient zum Starten des Imports der Backup-Daten, welche in eine CSV-Datei exportiert wurden. Nach jeder Datenerhebung wurden alle Devices und Signale in CSV-Dateien exportiert und im Backup-Order abgespeichert. Damit kann der Server auf verschiedenen Geräten gestartet und Daten in die Datenbank importiert werden.

|  |  |
| --- | --- |
|  | Neues Device registrieren |
| URI | beacons/api/internal/devices |
| HTTP-Methode | POST |
| Body-Parameter | JSON-Objekt, z.B.:  {  "fingerPrint" : "OnePlus/Nord\_EEA/Nord:11/RP1A.201005.001… ",  "manufacturer" : "OnePlus",  "brand" : "OnePlus",  "model" : "AC2003",  "sdk" : "30"  } |

Tabelle 5: Devices-API: Neues Device registrieren

|  |  |
| --- | --- |
|  | Device-Backup importieren |
| URI | beacons/api/internal/devices/import\_backup |
| HTTP-Methode | POST |
| Body-Parameter | String: CSV-Input-Dateiname |

Tabelle 6: Devices-API: Device-Backup importieren

### Signal

Im Signal-Rest-Controller wurden drei neue Schnittstellen implementiert. Die erste Schnittstelle wird vom Client aufgerufen, um empfangene Signale zu übermitteln und in der Datenbank zu speichern. Die zweite Schnittstelle dient zum Starten des Imports der Backup-Daten, welche in eine CSV-Datei exportiert wurden. Nach jeder Datenerhebung wurden alle Devices und Signale in CSV-Dateien exportiert und zur Backup-Order abgespeichert. Damit kann der Server auf verschiedenen Geräten gestartet und Daten in die Datenbank importiert werden. Die dritte Schnittstelle verknüpft alle existierenden Signale mit den existierenden Beacons, basierend auf den Major- und Minor-Kennungen. Diese API wurde für die Analyse des Beacon-Setups verwendet.

|  |  |
| --- | --- |
|  | Signale speichern |
| URI | beacons/api/internal/signals |
| HTTP-Methode | POST |
| Body-Parameter | JSON-Array, z.B.:  [{  "signalTimestamp" : "2022-05-11 12:00:00.000",  "rssi" : "-70",  "uuid" : "2686f39c-bada-4658-854a-a62e7e5e8b8d",  "major" : "123",  "minor" : "4564"  }, {…} ] |

Tabelle 7: Signals-API: Signale speichern

|  |  |
| --- | --- |
|  | Signal-Backup importieren |
| URI | beacons/api/internal/signals/import\_backup |
| HTTP-Methode | POST |
| Body-Parameter | String: CSV-Inpufile-Name |

Tabelle 8: Signals-API: Signal-Backup importieren

|  |  |
| --- | --- |
|  | Signale mit Beacons verknüpfen |
| URI | beacons/api/internal/signals/match\_beacons |
| HTTP-Methode | POST |

Tabelle 9: Signals-API: Signale mit Beacons verknüpfen

### Route

Mit dem Route-Rest-Controller wurden zwei neue Schnittstellen implementiert. Die erste Schnittstelle dient zur Ausführung der Routenberechnung für den in den URI-Parametern bestimmten Zeitraum. Dabei werden separate Routen für alle Devices im gegebenen Zeitraum berechnet. Der implementierte Prozess zur Signalauswertung und Routenberechnung wird im Kapitel 4.2.6 genauer vorgestellt. Direkt nach der Berechnung werden die Routen, in einer mit Opendatasoft kompatiblen Struktur, in CSV-Dateien exportiert. Der Parameter type bestimmt den Typ des Exports. Wenn point als Typ ausgewählt wird, wird die Route in Opendatasoft als Geopunkte dargestellt. Wenn line als Typ ausgewählt wird, werden die Punkte mit Linien verbunden. Die zweite Schnittstelle exportiert bereits berechnete Route in CSV-Dateien.

|  |  |
| --- | --- |
|  | Routen berechnen |
| URI | beacons/api/internal/routes/calculate |
| HTTP-Methode | GET |
| URI-Parameter | String: routeStart (Format: yyyy-MM-dd HH:mm:ss)  String: routeEnd (Format: yyyy-MM-dd HH:mm:ss)  String: type (point/line) |

Tabelle 10: Routes-API: Routen berechnen

|  |  |
| --- | --- |
|  | Route exportieren |
| URI | beacons/api/internal/signals/import\_backup |
| HTTP-Methode | GET |
| URI-Parameter | Long: routeId  String: type (point/line) |

Tabelle 11: Routes-API: Route exportieren

### Routenberechnung

Die Routenberechnung ist in drei Unterprozesse aufgeteilt. Dabei wird jeder dieser drei Unterprozesse für jedes im System registrierte Device ausgeführt, um separate Routen zu berechnen. Abbildung 6 zeigt das Aktivitätsdiagramm für die Routenberechnung. Zuerst werden die Signale für den über die REST-Schnittstellenparameter definierten Zeitraum aus der Datenbank gelesen. Danach werden diese Signale aufbereitet und vorgefiltert. Basierend auf den aufbereiteten und vorgefilterten Signalen werden die Routen berechnet.

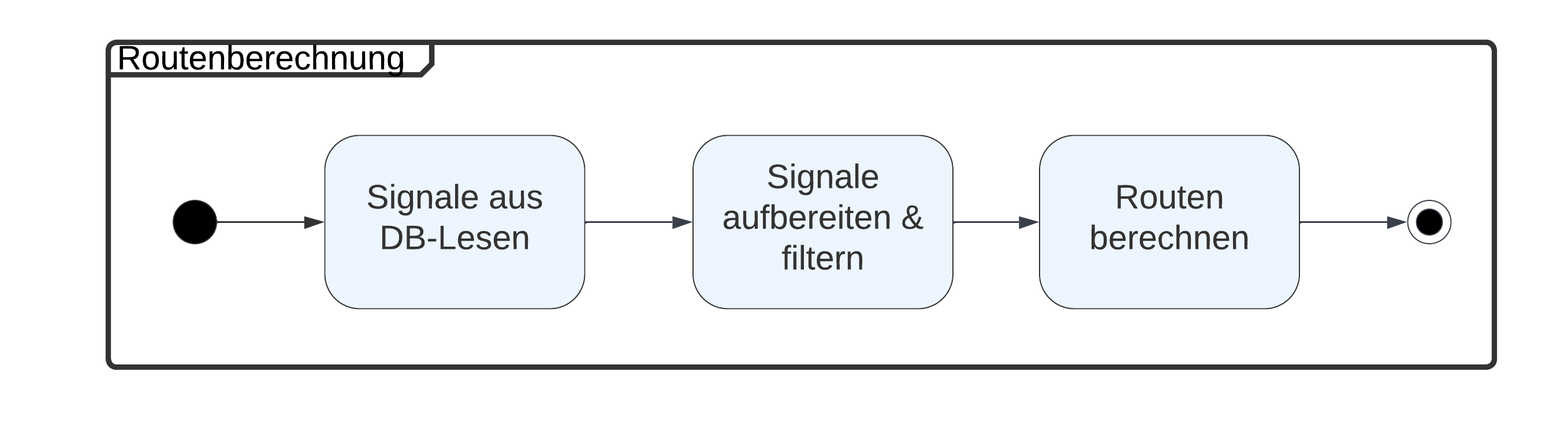


Abbildung 6: Aktivitätsdiagramm der Routenberechnung

Das Aktivitätsdiagramm in Abbildung 7 demonstriert den Aufbereitungs- und Filterungsprozess der Signale. Zuerst werden die Distanzen berechnet und das Signal wird, vorausgesetzt es stammt von einem der SBB-Beacons, mit dem entsprechenden Beacon verknüpf. Für jedes Signal werden dabei zwei Distanzen mit der Distanzberechnungsformell**Fehler! Verweisquelle konnte nicht gefunden werden.** berechnet und gespeichert. Die erste Distanz basiert nur auf RSSI-Wert des einen Signals und wird im Folgenden als Simple-RSSI-Distanz referenziert. Die zweite Distanz basiert auf dem gleitenden Durchschnitt der RSSI der Signale vom selben Beacon über 20 Sekunden und wird im Folgenden als Sliding-Window-Distanz referenziert. Jedes Signal wird auch nach dem Stockwerk des Standorts des Beacons und der Mindestdistanz des Signals gefiltert. Zuletzt wird ein Filter angewendet, der überprüft, ob eine Mindestanzahl an Signalen des gleichen Beacons innerhalb eines bestimmten Zeitraums erkannt wurde. Die Parameter der Filter können in den Application-Properties konfiguriert werden und sind im Kapitel 4.4 dokumentiert

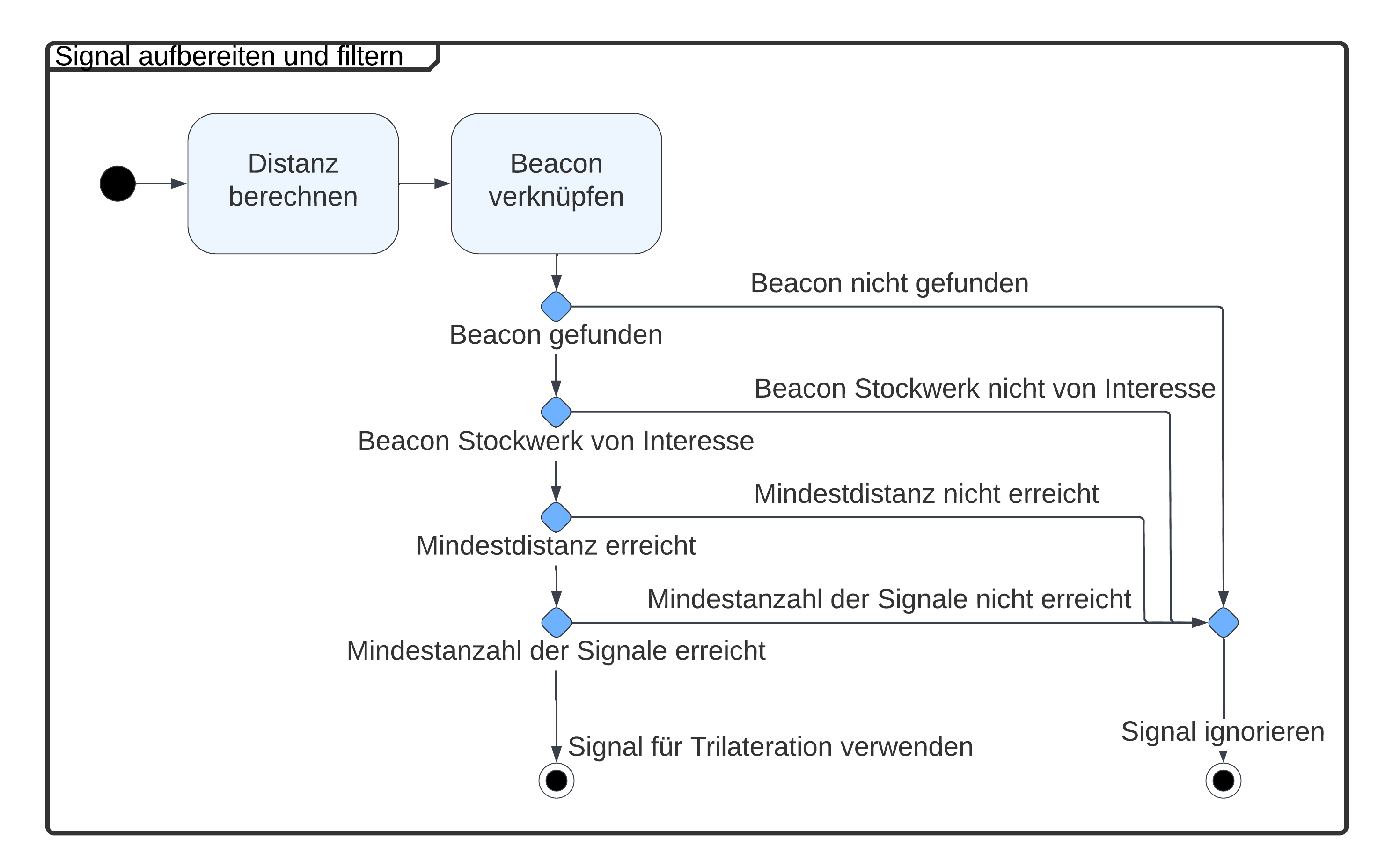


Abbildung 7: Aktivitätsdiagramm der Signalaufbereitung und -filterung

Basierend auf den vorgefilterten Signalen werden für den gleichen Zeitraum pro Device zwei separate Routen berechnet. Die erste Route wird basierend auf den Simple-RSSI-Distanzen berechnet. Die Berechnung der zweiten Routen basiert auf den Sliding-Window-Distanzen. Zu Beginn der Routenberechnung werden die vorgefilterten Signale in Triples aggregiert. Jedes Triple beinhaltet die drei Signale mit den kürzesten Distanzen, welche zum gleichen Zeitpunkt empfangen wurden. Für jedes Triple wird basierend auf den Distanzen der Signale und den Koordinaten der zugehörigen Beacons eine Position, mit dem im Kapitel 2 vorgestellten Trilaterationsverfahren, berechnet.

Das Trilaterationsverfahren wurde mittels vier verschiedenen Berechnungsmethoden implementiert und in der Testklasse TrilaterationAlgorithmTest.java mittels Testdaten wie auch mittels erhobener Daten verglichen. In Tabelle 12 sind die vier Klassen mit den unterschiedlichen Berechnungsmethoden beschrieben. Die besten Ergebnisse lieferte die zweite Berechnungsmethode, da die Positionen besser approximiert wurden.

|  |  |  |
| --- | --- | --- |
| Nr. | Klasse | Beschreibung |
| 1 | **TrilaterationLibraryAlgorithm.java** | Trilateration mit einer existierender Trilaterations-Library von lemmingapex (2019) |
| 2 | **StackoverflowTrilaterationAlgorithm.java** | Trilateration basierend auf einem von Ruddel (2017) auf Stackoverflow vorgeschlagenen Algorithmus. |
| 3 | **TrilaterationAlgorithm.java** | Trilateration basierend auf einem von Cannizzaro et al. (2019, S. 6) vorgeschlagenem Algorithmus. |
| 4 | **CellPhoneTrilaterationAlgorithm.java** | Trilateration basierend auf einem auf 101Computing (2019) vorgeschlagenen Algorithmus. |

Tabelle 12: Implementierte Trilaterationsmethoden

## Architektur

Das Packagediagramm in Abbildung 8 zeigt die Server-Architektur exemplarisch auf. Die höchste Ebene ist der Service Layer und umfasst REST-Schnittstellen für die Kommunikation mit externen Systemen, wie dem Android-Client. Der Business Layer beinhaltet die gesamte Applikationslogik, wie zum Beispiel die Signalauswertung und Routenberechnung. Der Data Layer ist für die Repräsentation der Daten und Kommunikation mit der Datenbank zuständig. Das Package .security beinhaltet die Implementation der JWT-Autorisierung und die REST-Security-Konfiguration. Im .error Package befindet sich die Implementation des zentralen Exception-Handlings. Das .utils Package beinhaltet allgemeine Helfer-Klassen, wie zum Beispiel einen BigDecimal-Kalkulator oder Enums.

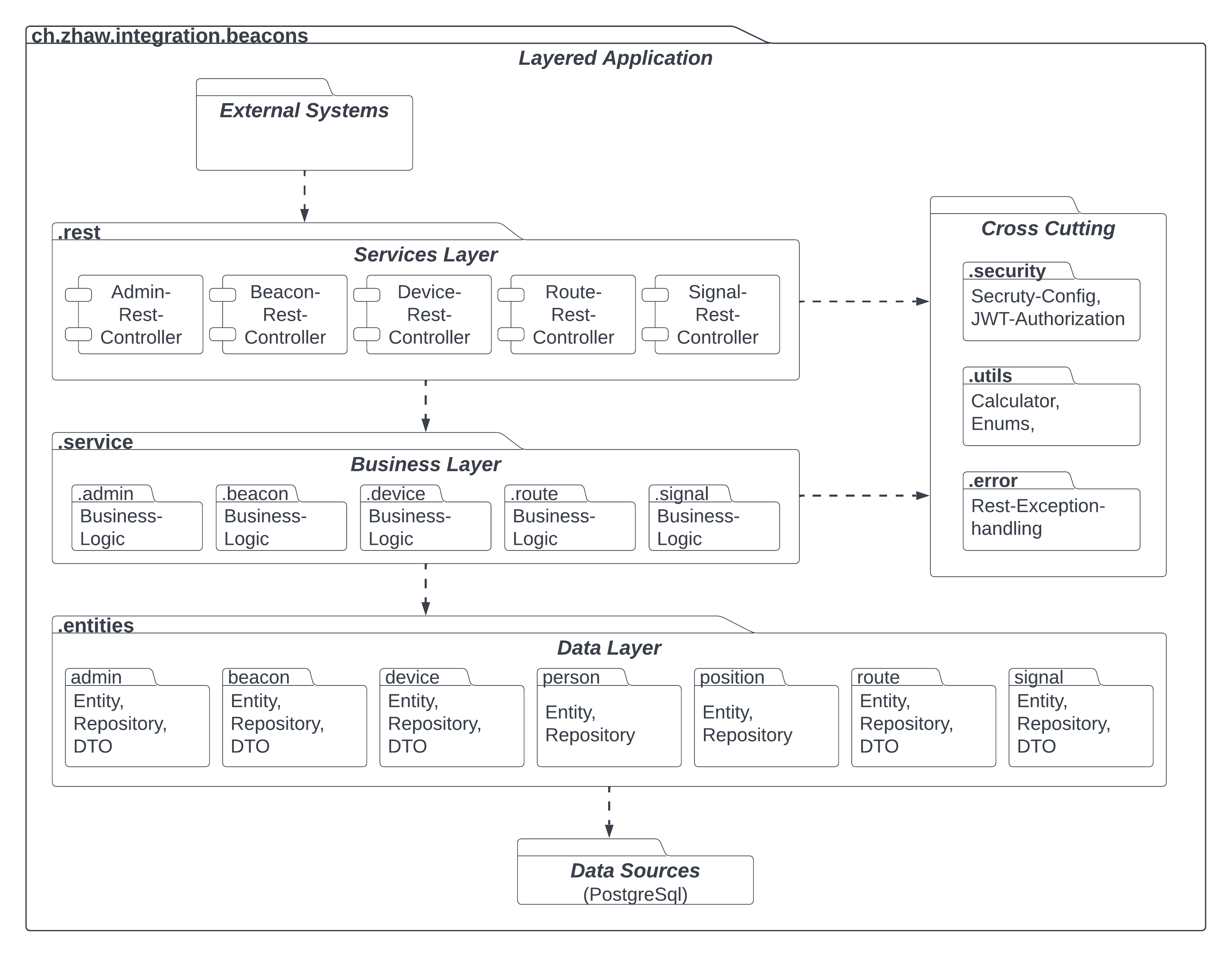


Abbildung 8: Packagediagramm der Server-Architektur

## Konfigurierbare Filter- und Berechnungsparameter

Die Datei application.properties beinhaltet verschiedene Konfigurationen wie zum Beispiel die Datenbankverbindung, JWT-Sicherheitseinstellungen sowie Berechnungs- und Filterungsparameter. In Tabelle 13 werden die konfigurierbaren Berechnungs- und Filterungsparameter beschrieben.

|  |  |
| --- | --- |
| Parameter | beacons.trilateration.signals.ignore.floor |
| Datentyp: Default-Wert | String-Array: 0, -1, -2 |
| Beschreibung | Definiert die Stockwerke, von welchen die Beacon und ihre Signale rausgefiltert werden. |
| Name | **beacons.trilateration.signal.min.count** |
| Datentyp: Default-Wert | Integer: 2 |
| Beschreibung | Definiert die Mindestanzahl von Signalen eines einzelnen Beacons welche innerhalb einer Periode erreicht werden muss, damit die Signale dieses Beacons für die Berechnungen berücksichtigt werden. |
| Parameter | **beacons.trilateration.signal.min.count.period** |
| Datentyp: Default-Wert | Integer: 10 |
| Beschreibung | Definiert die Periode in Sekunden, in welcher eine Mindestanzahl von Signalen eines einzelnen Beacons erreicht werden muss. |
| Parameter | **beacons.trilateration.beacon.min.distance** |
| Datentyp: Default-Wert | Double: 1.0 |
| Beschreibung | Definiert die Mindestdistanz eines Signals welche erreicht werden muss, damit ein Signal für die Berechnung in Betracht gezogen wird. Dient zum Rausfiltern falscher Signale, da Distanz zu einem Beacon von weniger als einem Meter aufgrund des Setups ausgeschlossen werden kann. |
| Parameter | **beacons.trilateration.distance.calculation.environmental.factor** |
| Datentyp: Default-Wert | Double: 1.5 |
| Beschreibung | Definiert die N-Konstante der Formel 1 zur Distanzberechnung. |
| Parameter | **beacons.calculation.bigdecimal.rounding.precision** |
| Datentyp: Default-Wert | Integer: 30 |
| Beschreibung | Definiert die Nachkomastelle an welcher Berechnete BigDecimal-Werte gerundet werden. |

Tabelle 13: Konfigurierbare Filter- und Berechnungsparameter