Hochschule für Technik und Wirtschaft Dresden
Fakultät Informatik/Mathematik

Diplomarbeit

im Studiengang Medieninformatik

Entwurf eines generischen Benachrichtigungssystems für Individualanwendungen

eingereicht von: Sabine Klose

eingereicht am: 06.05.2025

Betreuer: Prof. Dr.-Ing. Jörg Vogt, Dipl.-Ing. Bastian Buder

Inhaltsverzeichnis

Abkürzungsverzeichnis

ANH Azure Notification Hubs

API Application Programming Interface

APNs Apple Push Notification Service

App Center Microsoft Visual Studio App Center

FCM Firebase Cloud Messaging

HTTPS Hypertext Transfer Protocol Secure

IoT Internet of Things

JPush JPush by Jiguang

LoRaWAN Long Range Wide Area Network

MQTT Message Queuing Telemetry Transport

n.g. nicht gemessen

n.r. nicht relevant

PhoneGap Push PhoneGap Push Plugin

PushBackend Backend der Push WebApp der Develappers GmbH

SDK Software Development Kit

WLAN Wireless Local Area Network

Glossar

API Eine API ist eine Anwendungsprogrammierschnittstelle. Genutzt werden APIs, um Drittprogramme auf Quelltext-Ebene an eine Software anzubinden. Der Datenaustausch via API erfolgt in Echtzeit (Özdil 2025).

API-Token Um über eine API auf ein Programm oder eine Anwendung zuzugreifen und den Nutzer zu identifizieren, ist in manchen Fällen ein API-Token nötig. Ein API-Token ist eine alphanumerische Zeichenfolge (aws 2025). Mit diesem Token erkennt die API berechtigte Nutzerinnen und Nutzer und schützt Programme und Systeme vor Fremdzugriff (IONOS 2023).

Apple Push Notification Service Der Apple Push Notification Service (APNs) ist ein Dienst, der die Zustellung von Benachrichtigungen an iOS-Geräte von Nutzern ermöglicht (Apple Developer 2025).

Cloud Die Cloud bezeichnet über das Internet zugängliche Serverinfrastrukturen mit darauf betriebenen Anwendungen und Datenbanken. Sie sind weltweit in Rechenzentren verteilt und ermöglichen es, auf eigene Server oder lokal installierte Software weitgehend zu verzichten (Cloudflare 2025).

Datenübertragungsrate Die Datenübertragungsrate ist die Menge an digitalen Daten, die in einer bestimmten Zeit von einem Ort zum anderen übertragen wird (Yasar 2022).

FirebaseApp Die Firebase App ist der Einstiegspunkt der Firebase SDKs. Sie enthält die allgemeine Konfiguration und den Status für Firebase-APIs (Firebase 2023).

GitHub GitHub ist eine cloudbasierte Plattform zur Speicherung, Freigabe und Zusammenarbeit an Code. Sie nutzt Git, ein Versionskontrollsystem, das Änderungen an Dateien effizient verfolgt, besonders bei gleichzeitigen Bearbeitungen durch mehrere Personen (GitHub 2025).

HTTPS HTTPS ist ein Kommunikationsprotokoll zur verschlüsselten Übertragung von Informationen zwischen Computern im World Wide Web (Microsoft 2014). Es schützt die Kommunikation zwischen Client und Server durch Verschlüsselung und gewährleistet die Integrität sowie Vertraulichkeit der Daten (Timonera 2023). Damit bildet es die Grundlage für die sichere Nutzung von REST-APIs (Timonera 2023).

Integrationsplattform Eine Integrationsplattform ist eine Softwarelösung, mit der die verschiedenen Systeme, Anwendungen und Datenquellen innerhalb einer

Organisation miteinander verbunden sind (Kärkkäinen 2025). In Smart-Home-Systemen steuert sie unter anderem, wie Sensordaten verarbeitet und weitergeleitet werden.

IoT Unter dem Internet der Dinge (Internet of Things) versteht man ein Netzwerk vernetzter Geräte, die Daten sowohl untereinander als auch mit der Cloud austauschen (Yasar u.a. 2024). IoT-Geräte verfügen in der Regel über Sensoren und Software und schließen sowohl mechanische und digitale Maschinen als auch Konsumgüter ein (Yasar u.a. 2024).

LoRaWAN LoRaWAN ist ein Schichtprotokoll, das speziell für großflächige öffentliche Netzwerke mit einem zentralen Betreiber konzipiert wurde (LoRaWAN 2025). Es ermöglicht energieeffiziente Kommunikation mit niedriger Datenrate über weite Distanzen innerhalb des abgedeckten Gebiets (Haxhibeqiri u. a. 2018).

Postman Postman ist eine weit verbreitete Plattform, die Entwicklern eine umfassende Umgebung für die Erstellung, Verwaltung und das Testen von APIs bietet. Sie ermöglicht es, API-Anfragen zu entwickeln, zu simulieren und zu testen (Postman 2025).

Postman Runner Der Postman Runner ermöglicht das Ausführen von API-Anfragen aus einer Sammlung mit verschiedenen Einstellungsmöglichkeiten. Er protokolliert Testergebnisse und übergibt Daten zwischen Anfragen (Postman 2024).

Software Development Kit Ein Software Development Kit (SDK) ist ein Satz von Software-Tools und Programmen, die von Hardware- und Softwareanbietern bereitgestellt werden und mit denen Entwickler Anwendungen für bestimmte Plattformen erstellen können (Yasar u.a. 2022). SDKs helfen Entwicklern, ihre Anwendungen einfach in die Dienste eines Anbieters zu integrieren (Yasar u.a. 2022).

Z-Wave Das Z-Wave-Protokoll ist eine interoperable, drahtlose, RF-basierte Kommunikationstechnologie, die speziell für Steuerungs-, Überwachungs- und Statusanzeigeanwendungen in Wohn- und Gewerbeumgebungen entwickelt wurde (Z-WAVE Alliance 2025).

Zigbee Zigbee ist eine standardbasierte drahtlose Technologie, die entwickelt wurde, um kostengünstige, stromsparende drahtlose Machine-to-Machine- und IoT-Netzwerke zu ermöglichen (Rosencrance 2017).

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Mit der fortschreitenden Digitalisierung verändert sich die Interaktion zwischen Menschen und Software grundlegend. Push-Benachrichtigungen haben sich dabei als zentrales Mittel etabliert, um kontextbezogene Informationen in Echtzeit zu übermitteln.

1.1 Motivation

Die zunehmende Verfügbarkeit umfangreicher Echtzeitdatenströme, insbesondere aus Sensor- und IoT-Systemen, eröffnet neue Möglichkeiten für die Entwicklung intelligenter, kontextsensitiver Anwendungen. Die proaktive Bereitstellung relevanter Informationen zum passenden Zeitpunkt trägt nicht nur zur Steigerung der Nutzerzufriedenheit bei, sondern ermöglicht zugleich eine effizientere Ressourcennutzung und Prozessoptimierung. Damit einhergehend wächst der Bedarf an flexiblen und modularen Architekturen, die eine zuverlässige Verarbeitung sowie eine gezielte Weiterleitung von Ereignissen in Echtzeit unterstützen. Bislang existiert jedoch kein universell einsetzbares System, das Push-Benachrichtigungen in eine individuelle Anwendung überträgt und gleichzeitig Funktionen zur Verwaltung von Nutzern und Endgeräten bereitstellt.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Konzeption und prototypische Umsetzung eines generischen Benachrichtigungssystems, das sich flexibel in Individualanwendungen integrieren lässt und auf sensorbasierte Ereignisse in Echtzeit reagieren kann. Im Mittelpunkt steht die Entwicklung eines modularen und erweiterbaren Systems, das unabhängig von spezifischen Sensorarten oder Plattformen arbeitet und sich an unterschiedliche Anwendungsfälle anpassen lässt. Zur Evaluation werden zuvor aufgestellte funktionale und nichtfunktionale Anforderungen geprüft. Ergänzend wird untersucht, inwieweit ein vergleichbares System ohne die Abhängigkeit von Drittanbieterdiensten realisiert werden kann.

2 Grundlagen

Ein Benachrichtigungssystem ist eine technische Infrastruktur, die dazu dient, Nutzer zeitnah über relevante Ereignisse oder Informationen zu informieren. In modernen Anwendungen spielt es eine zentrale Rolle, indem es die Kommunikation zwischen System und Nutzer optimiert und die Interaktivität erhöht. Ein universelles Benachrichtigungssystem, das verschiedene Kommunikationsprotokolle integriert und plattformübergreifend Nachrichten versendet, schafft eine zentrale Verbindung zwischen Sensoren und Anwendungen (siehe Abb. 2.1).

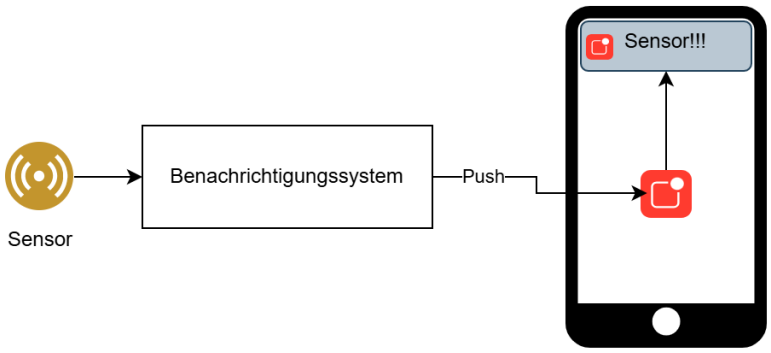

Abbildung 2.1: Generisches Benachrichtigungssystem.

Push-Benachrichtigungen sind ein zentraler Bestandteil moderner Kommunikationssysteme, da sie es ermöglichen, Informationen gezielt und ohne vorherige Nutzeranfrage zu übermitteln (Eka Putra u. a. 2024). Im Gegensatz zu Pull-Mechanismen, bei denen der Client in regelmäßigen Abständen Anfragen stellt, reagieren Push-Systeme unmittelbar auf konkrete Ereignisse. Push-Benachrichtigungstechnologien erlauben die sofortige Übermittlung von Informationen an die Benutzer (Eka Putra u. a. 2024), indem sie es ermöglichen, Daten vom Server aktiv an das Gerät des Nutzers zu senden, ohne dass eine explizite Anfrage des Geräts erforderlich ist. Zwar können Pull-basierte Verfahren im Vergleich energieeffizienter sein, sie verursachen jedoch höhere Latenzzeiten (Burgstahler u. a. 2014). Dies stellt insbesondere in Anwendungen, die schnelle Reaktionszeiten erfordern, einen entscheidenden Nachteil dar. Die ereignisorientierte Architektur von Push-Benachrichtigungen erlaubt ein dynamisches und reaktives Systemverhalten, das vor allem in zeitkritischen Szenarien wie Alarmierungen oder Statusmeldungen von hoher Bedeutung ist. Eine zentrale Rolle nehmen dabei Sensoren ein, die als Quellen relevanter Ereignisse fungieren, indem sie physikalische Veränderungen wie Bewegung oder Temperatur erfassen und automatisiert entsprechende Folgeaktionen auslösen, beispielsweise das Versenden

einer Push-Nachricht.

Individualanwendungen sind speziell auf die Bedürfnisse eines bestimmten Anwenderkreises oder Kunden zugeschnittene Softwarelösungen. Sie unterscheiden sich von Standardsoftware durch ihre Anpassungsfähigkeit an unternehmensspezifische Prozesse, Strukturen und Anforderungen. Beispielsweise in der industriellen Fertigung, im Smart-Home-Umfeld oder im Gesundheitswesen, bieten Individualanwendungen die Flexibilität, spezifische Sensorik und Systemlogiken zu berücksichtigen.

3 Anforderungsanalyse

Die Anforderungsanalyse bildet einen entscheidenden Schritt in der Softwareentwicklung, da sie die Grundlage für die spätere Umsetzung eines Systems schafft.

In diesem Kapitel werden die spezifischen Anforderungen an ein generisches Benachrichtigungssystem für Individualanwendungen erarbeitet. Dabei werden zunächst die Anwendungsfälle definiert, die Zielgruppen genannt und die funktionalen und nichtfunktionalen Anforderungen aufgestellt. Ziel ist es, eine fundierte Basis für die Systemarchitektur zu schaffen und spätere Entwicklungsfehler sowie hohe Kosten zu vermeiden, da es praktisch nicht möglich ist, mangelhafte Ergebnisse einer Anforderungsanalyse durch besondere Qualität in späteren Phasen wieder auszugleichen (Kleuker 2013).

3.1 Anwendungsfälle

Sensorbasierte Push-Benachrichtigungen bieten vielseitige Anwendungsmöglichkeiten, da sie unmittelbar auf Ereignisse reagieren können, die durch Sensoren erfasst werden.

Im Smart-Home-Bereich senden beispielsweise Fenster- und Türsensoren Benachrichtigungen, wenn ein Fenster geöffnet wird, während Feuermelder Alarm schlagen, sobald Feuer oder Rauch erkannt wird (vgl. Saeed u. a. 2018). Bewegungssensoren können zudem unbefugten Zutritt melden und so zur Sicherheit beitragen.
Auch in der Industrie spielen Sensoren eine wichtige Rolle, insbesondere bei der Überwachung von Maschinen und Anlagen (vgl. Ganiger u. a. 2024). Sie registrieren Überhitzung, ungewöhnliche Vibrationen oder Ausfälle und lösen entsprechende Push-Nachrichten aus, um Ausfallzeiten zu reduzieren.
Im Gesundheitswesen helfen medizinische Sensoren dabei, kritische Zustände frühzeitig zu erkennen. Sie messen Vitalparameter wie Herzfrequenz, Blutdruck oder Blutzuckerwerte und informieren Ärzte oder Pflegepersonal bei auffälligen Abweichungen. Außerdem können kritische Laborwerte direkt an Mitarbeitende weitergeleitet werden (vgl. Adhyaru u. a. 2023).
Auch in der Umweltüberwachung kommen Sensoren zum Einsatz, indem sie beispielsweise Luftqualität, Temperatur oder Wasserstände messen und bei kritischen Werten Warnungen ausgeben, etwa bei Überschwemmungen oder erhöhter Schadstoffbelastung (vgl. Halvorsen u. a. 2018).

Durch den gezielten Einsatz sensorbasierter Push-Benachrichtigungen lassen sich Prozesse optimieren, Sicherheit erhöhen und Effizienz in zahlreichen Anwendungsbereichen verbessern.

3.2 Zielgruppen

Aufgrund der vielfältigen Anwendungsfälle des Benachrichtigungssystems lassen sich verschiedene Zielgruppen identifizieren, die von dessen Einsatz profitieren.

Privatanwender im Bereich Smart Home stellen eine zentrale Zielgruppe für sensorbasierte Benachrichtigungssysteme dar. Sie profitieren von erhöhter Sicherheit und gesteigertem Komfort in ihrem Zuhause. Dazu zählen insbesondere Hausbesitzer, Mieter und technikaffine Personen, die ihre Wohnumgebung smarter und effizienter gestalten möchten.

Eine weitere bedeutende Zielgruppe sind Hersteller von IoT-Geräten und Sensoren. Diese Unternehmen entwickeln innovative Technologien und können die Funktionalität ihrer Produkte durch die Integration von Push-Benachrichtigungen erweitern. Beispiele hierfür sind Produzenten von Rauchmeldern, Bewegungssensoren, Smart-Home-Geräten, Wearables sowie Systemen zur industriellen Überwachung.

Außerdem zählen auch Unternehmen, insbesondere aus der Industrie, zu den relevanten Zielgruppen. Durch den Einsatz sensorbasierter Benachrichtigungssysteme können sie Maschinen und Anlagen überwachen, Wartungsbedarfe frühzeitig erkennen und Ausfallzeiten minimieren. Dies führt zu einer erhöhten Betriebseffizienz und Kosteneinsparungen, insbesondere in Bereichen wie der Fertigung, Logistik und Prozessautomatisierung.

Tabelle 3.1: Zielgruppen des generischen Benachrichtigungssystems

Zielgruppe	**Beschreibung**
Privatanwender Smart-Home-Bereich	Nutzer von Smart-Home-Systemen, die Sicherheit und Komfort in ihrem Zuhause erhöhen möchten. Beispiele: Hausbesitzer, Mieter, technikaffine Nutzer.
Hersteller von IoT-Geräten und Sensoren	Unternehmen, die intelligente Geräte und Sensoren entwickeln und vermarkten und durch Push-Benachrichtigungen die Funktionalität ihrer Produkte erweitern möchten. Beispiele: Hersteller von Rauchmeldern, Bewegungssensoren, Smart-Home-Geräten, Wearables, industriellen Überwachungssystemen.

Zielgruppe	Beschreibung
Unternehmen zur Maschinenüberwachung	Industrieunternehmen, die sensorbasierte Systeme zur Echtzeitüberwachung von Maschinen und Anlagen nutzen, um Wartungsbedarfe frühzeitig zu erkennen und Ausfallzeiten zu minimieren. Beispiele: Unternehmen aus der Fertigung, Logistik, Prozessautomatisierung.

3.3 Anforderungen

Im Abschnitt Anforderungen werden die zentralen Anforderungen an das zu entwickelnde System beschrieben. Sie sind entscheidend für die erfolgreiche Umsetzung und Integration des generischen Benachrichtigungssystems. Dieses System soll Push-Benachrichtigungen auf Basis von Sensordaten versenden. Dabei werden die Anforderungen in zwei Kategorien unterteilt: funktionale und nichtfunktionale Anforderungen.

3.3.1 Funktionale Anforderungen

Eine funktionale Anforderung legt eine vom Softwaresystem oder einer seiner Komponenten bereitzustellende Funktion oder bereitzustellenden Service fest (Balzert 2009).

FA 1 Verarbeitung der Sensordaten

(a) Das System soll eine flexible Architektur bereitstellen, die Regelung und Automatisierung ermöglicht. Es muss in der Lage sein, definierte Ereignisse zu erkennen und entsprechende Aktionen auszuführen.

(b) Das System muss prüfen, ob auf Basis der Sensordaten eine Benachrichtigung ausgelöst werden soll und diese als Push-Benachrichtigung gegebenenfalls an den Nutzer senden.

(c) Die Benachrichtigung muss einen Titel, einen Text und den Zeitstempel des Ereignisses enthalten.

FA 2 Speicherung der Benachrichtigungen

(a) Alle gesendeten Benachrichtigungen müssen gespeichert werden und abrufbar sein, um dem Nutzer eine vollständige Historie zu ermöglichen.

FA 3 Mandantenfähigkeit

(a) Das System muss mehrere unabhängige Mandanten verwalten können. Dabei muss es sämtliche Daten und Konfigurationen jedes Mandanten isoliert behandeln, jedoch soll die gleiche Infrastruktur genutzt werden.

FA 4 Geräteverwaltung

(a) Das System muss die Registrierung, Verwaltung und Löschung der Endgeräte des Nutzers ermöglichen, die Push-Benachrichtigungen empfangen.

(b) Jedes Endgerät muss eindeutig identifizierbar sein und einem Nutzer zugeordnet werden, um gezielt Benachrichtigungen zu senden.

FA 5 Integration in Anwendungen

(a) Die Anwendung muss empfangene Push-Benachrichtigungen zuverlässig verarbeiten und an den Endbenutzer weiterleiten, unabhängig davon, ob die Anwendung aktiv oder im Hintergrund ausgeführt wird.

(b) Die Integration in individuelle Anwendungen muss über ein Software Development Kit (SDK) oder eine Application Programming Interface (API) ermöglicht werden, um das Empfangen sowie die Anzeige und Verwaltung von Push-Benachrichtigungen sicherzustellen.

3.3.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen definieren die allgemeinen Merkmale und das Verhalten eines Systems. Letztlich bestimmen sie die Qualität, die Leistung und die Benutzererfahrung des Systems (Subahi 2023).

NFA 1 Zuverlässigkeit

(a) Das System muss eine hohe Verfügbarkeit gewährleisten, indem es auch bei schwankender Netzqualität eine stabile und zuverlässige Zustellung von Benachrichtigungen sicherstellt.

NFA 2 Skalierbarkeit

(a) Das System muss in der Lage sein, mit bis zu 1 Mio gleichzeitigen Nutzern zu skalieren, ohne dass die Leistung deutlich beeinträchtigt wird.

(b) Es muss in der Lage sein, mindestens 500 Sensoren pro Nutzer parallel zu verarbeiten.

(c) Die zuverlässige Zustellung von Push-Benachrichtigungen mit einer Frequenz von zehn Nachrichten pro Minute und pro Benutzer muss sichergestellt sein.

NFA 3 Reaktionszeiten

(a) Das System muss in der Lage sein, Benachrichtigungen mit einer kurzen Reaktionszeit auszulösen, wobei die Verzögerung zwischen dem Erkennen eines Ereignisses und der Benachrichtigung möglichst gering gehalten werden muss.

(b) Das System muss in der Lage sein, Benachrichtigungen spätestens zehn Sekunden nach dem Erkennen eines Sensorereignisses auszulösen.

NFA 4 Wartbarkeit und Erweiterbarkeit

(a) Das System muss so entwickelt werden, dass es leicht gewartet und bei Bedarf erweitert werden kann, um neue Anforderungen oder Funktionen zu integrieren.

(b) Die Architektur sollte so gestaltet sein, dass Updates oder Änderungen ohne signifikante Unterbrechung des Dienstes vorgenommen werden können.

NFA 5 Ressourcennutzung

(a) Das System muss effizient mit Ressourcen wie Speicher und Rechenleistung umgehen.

(b) Der Energieverbrauch auf mobilen Endgeräten soll möglichst gering gehalten werden, um die Akkulaufzeit nicht unnötig zu beeinträchtigen.

NFA 6 Aktualität und Zukunftsfähigkeit

(a) Das System sollte auf Technologien basieren, die dem aktuellen Stand der Technik entsprechen und kontinuierlich weiterentwickelt werden. Nur so lässt sich eine nachhaltige Nutzbarkeit gewährleisten und gleichzeitig sicherstellen, dass neu entdeckte Sicherheitslücken zeitnah behoben werden können.

NFA 7 Externe Dienste

(a) Die Auswahl externer, kostenpflichtiger Dienste muss so erfolgen, dass die Kosten auch bei steigender Auslastung oder wachsender Nutzerzahl nicht unkontrolliert ansteigen. Es müssen nur solche Dienste gewählt werden, die eine kosteneffiziente Skalierung ermöglichen und bei hoher Last keine exponentiellen Kostensteigerungen verursachen.

(b) Externe Dienste müssen über eine Lizenz verfügen, die eine kommerzielle Nutzung nicht untersagt.

NFA 8 Datenschutz und Datensicherheit

(a) Das System muss die aktuellen Sicherheitsstandards einhalten, um den Schutz personenbezogener Daten und die Datensicherheit zu gewährleisten.

NFA 9 Benutzerfreundlichkeit

(a) Die Anwendung sollte sowohl für technisch versierte als auch weniger erfahrene Nutzer intuitiv bedienbar sein.

NFA 10 Kompatibilität

(a) Das System muss mit einer Vielzahl weit verbreiteter Sensoren und Kommunikationsprotokollen kompatibel sein, um eine hohe Flexibilität und Interoperabilität zu gewährleisten.

NFA 11 Portabilität

(a) Push-Benachrichtigungen müssen auf den Plattformen iOS, Android und Web bereitgestellt werden.

4 Stand der Technik

Im Abschnitt Stand der Technik werden bestehende Systeme untersucht und mit den zuvor definierten Anforderungen abgeglichen. Dabei wird die Funktionsweise dieser Systeme analysiert, um ihre Stärken und Schwächen im Hinblick auf die Anforderungen zu identifizieren. Ziel dieser Analyse ist es, die bestehenden Lösungen zu bewerten und gegebenenfalls Verbesserungspotenziale für die Entwicklung eines neuen Systems aufzuzeigen.
Es ist jedoch zu beachten, dass es keine universelle monolithische Lösung gibt, die sämtliche Anforderungen in vollem Umfang abdecken kann. Daher muss das geplante System aus mehreren modularen Komponenten bestehen, die jeweils auf bestimmte Anforderungen zugeschnitten sind und im Zusammenspiel die gewünschte Funktionalität und Effizienz bieten.

4.1 Vergleich von Integrationsplattformen

Es existiert eine Vielzahl an markenabhängigen IoT-Geräten und Sensoren, die über Home-Automation-Systeme wie Apple HomeKit, Google Home, Amazon Alexa oder Samsung SmartThings verwaltet werden können. Jedoch weisen diese Plattformen spezifische Einschränkungen auf, da sie in der Regel nur mit einer begrenzten Auswahl an kompatiblen Geräten genutzt werden können.

Zur Überwindung dieser Einschränkungen wurden herstellerunabhängige Plattformen entwickelt. Zu den derzeit am weitesten verbreiteten zählen ioBroker, openHAB und Home Assistant. Sie ermöglichen die zentrale Verwaltung unterschiedlichster Geräte und Kommunikationsprotokolle, indem sie eine Vielzahl von Integrationen und Erweiterungen bereitstellen.

Im weiteren Verlauf werden diese drei Plattformen einer eingehenden Analyse unterzogen und miteinander verglichen, um deren Potenziale, Anwendungsbereiche und Grenzen zu evaluieren. Diese Plattformen sollen die Informationen dann an ein bestehendes Benachrichtigungssystem oder ein externes Backend weiter leiten können (siehe Abb. 4.1, S. 11).

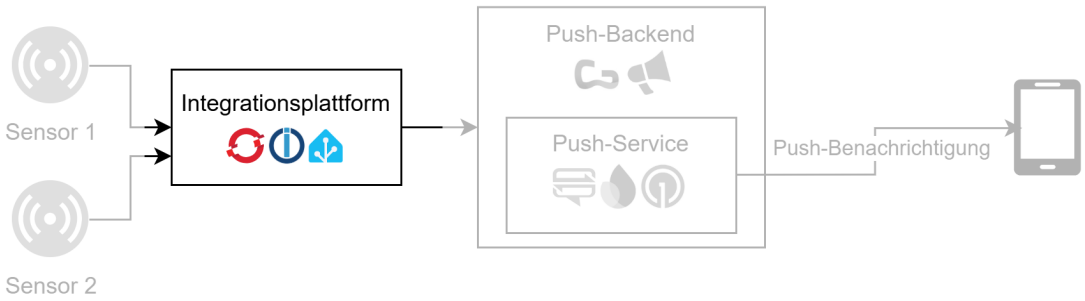

Abbildung 4.1: Integrationsplattformen in dem generischen Benachrichtigungssystem.

Da die Integrationsplattformen lediglich einen Teil des Gesamtsystems darstellen, sind nicht alle zuvor festgelegten funktionalen und nichtfunktionalen Anforderungen in diesem Kontext relevant. Insbesondere beziehen sich die Anforderungen in den Abschnitten FA 2, FA 3, FA 4, FA 5 sowie NFA 11 nicht auf die Plattformen (siehe Tab A.1, S. 101). Darüber hinaus liegen zu der in NFA 1 behandelten Zuverlässigkeit keine weiterführenden Informationen oder Tests vor, die eine tiefergehende Bewertung ermöglichen würden.

4.1.1 ioBroker

ioBroker wird als Integrationsplattform für das Internet der Dinge beschrieben (ioBroker 2024b) und ist in der Programmiersprache JavaScript entwickelt worden (ioBroker 2024b).

Verarbeitung der Sensordaten (FA 1)

Zur Unterstützung der Automatisierung lassen sich benutzerdefinierte Regeln definieren (FA 1a) (ioBroker 2024b). Darüber hinaus besteht die Möglichkeit, eigene Adapter in JavaScript oder TypeScript zu entwickeln (ioBroker 2024b). Diese können dazu verwendet werden, Nachrichten zu erstellen (FA 1c) und an das Backend zu übermitteln, das daraufhin Push-Benachrichtigungen auslöst (FA 1b).

Skalierbarkeit (NFA 2)

ioBroker macht keine konkreten Angaben zur Skalierbarkeit, wobei die maximale Anzahl an Sensoren von den verwendeten Adaptern abhängt.

Reaktionszeiten (NFA 3)

Es gibt ebenfalls keine konkreten Angaben zu der Fähigkeit von ioBroker, Sensordaten schnellstmöglich zu erkennen und zu verarbeiten. Jedoch hängt ebenfalls maßgeblich vom verwendeten Adapter ab.

Warbarkeit und Erweiterbarkeit (NFA 4)

Da ioBroker Open Source ist (ioBroker 2024a), bietet es die Möglichkeit zur individuellen Wartung und Modifikation, wodurch es flexibel an spezifische Anforderungen angepasst werden kann. Neue Adapter können jederzeit während des laufenden Betriebs hinzugefügt oder aktualisiert werden (ioBroker 2024b), wodurch ioBroker eine ausgezeichnete Erweiterbarkeit aufweist. Diese Eigenschaft ermöglicht es, das System dynamisch zu erweitern, ohne den laufenden Betrieb zu unterbrechen.

Ressourcenverbrauch (NFA 5)

Zu dem Ressourcenverbrauch macht ioBroker keine Angaben, jedoch wird dies von der Auslastung des Systems abhängen.

Aktualität und Zukunftsfähigkeit (NFA 6)

ioBroker wurde 2014 ins Leben gerufen. In den letzten Jahren hat die Aktivität rund um ioBroker weltweit abgenommen (siehe Abb. 4.2), was sich auch in einer geringeren Verbreitung außerhalb des deutschsprachigen Raums widerspiegelt. Ungefähr 91 % der Benutzer sind deutschsprachig (ioBroker 2025b)(Stand Januar 2025). Es gibt ein Forum sowie einen Discord-Server, in denen Nutzer Fragen stellen und Unterstützung erhalten können. Die Plattform wird kontinuierlich weiterentwickelt, wie an den Veröffentlichungen auf GitHub zu erkennen ist (Fischer 2025). Dies gewährleistet, dass die Software kontinuierlich dem aktuellen Stand der Technik entspricht und notwendige Fehlerkorrekturen zeitnah erfolgen.

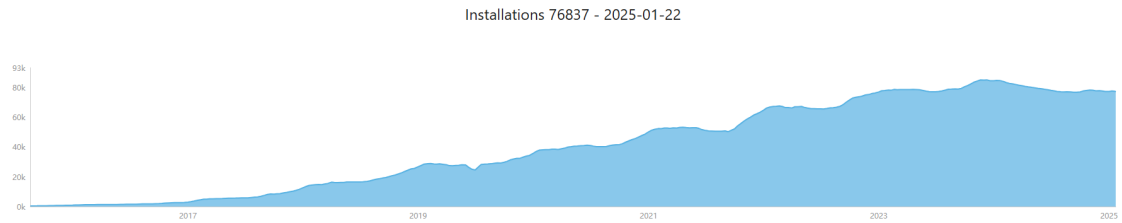

Abbildung 4.2: Nutzerstatistik basierend auf ioBroker-Installationen. (Quelle: ioBroker 2025b)

Kosten und Lizenz (NFA 7)

Als Open-Source-Projekt ist ioBroker unter der MIT-Lizenz lizenziert (Haev 2024b), was den freien Zugang und die Modifikation des Quellcodes ermöglicht. Diese Lizenzierung bietet nicht nur die freie Nutzung, sondern auch die Möglichkeit einer kommerziellen Anwendung der Plattform. Die Bedingung ist lediglich, dass ein Lizenz- und Urheberrechtshinweis eingefügt wird. Bei selbst gehostetem Betrieb fallen keine zusätzlichen Kosten an, außer die Kosten für die dafür benötigten Ressourcen. Die Nutzung der ioBroker-Cloud-Dienste oder spezieller Adapter kann jedoch mit zusätzlichen Gebühren verbunden sein.

Datenschutz (NFA 8)

Die Datenschutzbestimmungen von ioBroker hängen maßgeblich davon ab, wie die Plattform betrieben wird. Wenn die Software lokal bereitgestellt wird, verbleiben sämtliche Daten in der eigenen Umgebung, was eine vollständige Kontrolle über den Datenschutz ermöglicht. Die Nutzung der ioBroker-Cloud führt hingegen zu einer Übergabe der Daten an den Cloud-Anbieter, wobei hierzu keine spezifischen Angaben zum Datenschutz gemacht werden.

Benutzeroberfläche (NFA 9)

Die Benutzeroberfläche von ioBroker ist grundsätzlich funktional. Eine anpassbare Visualisierung ist über verschiedene Visualisierungsadapter möglich, die es den Nutzern erlauben, das Interface nach ihren individuellen Bedürfnissen zu gestalten. Allerdings sind einige dieser Visualisierungsadapter kommerziell kostenpflichtig.

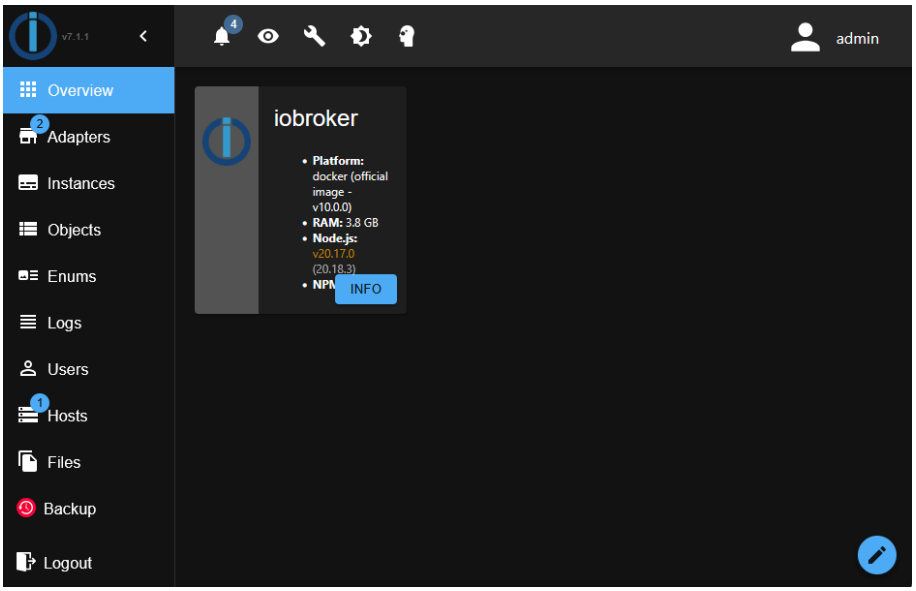

Abbildung 4.3: Standard Benutzeroberfläche ioBroker als Admin.

Kompatibilität (NFA 10)

Mit über 640 verfügbaren Adaptern (ioBroker 2025a) bietet die Plattform eine breite Palette an Erweiterungsmöglichkeiten, um die Funktionalitäten an die spezifischen Bedürfnisse der Nutzer anzupassen.

4.1.2 openHAB

openHAB verspricht, eine hersteller- und technologieunabhängige Open-Source-Plattform für die Hausautomatisierung zu sein (openHAB 2025f).

Verarbeitung der Sensordaten (FA 1)

Zur Realisierung automatisierter Abläufe können in openHAB benutzerdefinierte Regeln erstellt werden (FA 1a) (openHAB 2025h). Darüber hinaus ermöglicht die Plattform die Entwicklung eigener Bindings in Java (openHAB 2025c), welche eine Anbindung externer Geräte oder Dienste realisieren (openHAB 2025c). Diese Bindings können verwendet werden, um Nachrichten zu generieren (FA 1c) und an ein zentrales Backend zu übertragen, das anschließend die Zustellung von Push-Benachrichtigungen übernimmt (FA 1b).

Skalierbarkeit (NFA 2)

openHAB liefert keine spezifischen Informationen zur Skalierbarkeit. Die Anzahl der unterstützten Sensoren kann jedoch von den eingesetzten Bindings abhängen.

Reaktionszeiten (NFA 3)

Die Geschwindigkeit der Verarbeitung von Sensordaten hängt von den eingesetzten Bindings und der zugrunde liegenden Infrastruktur ab. Weitere Angaben werden von OpenHAB dazu nicht gemacht.

Wartbarkeit und Erweiterbarkeit (NFA 4)

Da openHAB eine Open-Source-Software ist (openHAB 2025f), kann es flexibel gewartet und modifiziert werden, um spezifischen Anforderungen gerecht zu werden. Weitere Bindings oder Add-ons können jederzeit hinzugefügt werden (openHAB 2025e), wodurch das System gut erweiterbar ist.

Ressourcenverbrauch (NFA 5)

Konkrete Angaben zum Ressourcenverbrauch liegen nicht vor. Die tatsächliche Auslastung hängt daher maßgeblich vom Umfang der jeweiligen Systemkonfiguration und der Anzahl aktiver Komponenten ab.

Aktualität und Zukunftsfähigkeit (NFA 6)

Die Plattform wurde im Jahr 2010 von Kai Kreuzer initiiert (WERTGARANTIE 2025) und verfügt über eine aktive Community mit einem eigenen Forum, in dem Benutzer Unterstützung erhalten können (openHAB 2025g). Die Plattform wird kontinuierlich weiterentwickelt, und etwa alle sechs Monate wird eine neue Minor-Version veröffentlicht (openHAB 2025d), was auf eine aktive und nachhaltige Pflege hinweist.

Kosten und Lizenz (NFA 7)

Lizenziert unter der Eclipse Public License 2.0 (Born u. a. 2019) erlaubt die Nutzung der Plattform auch im kommerziellen Kontext. Voraussetzung dafür ist die Offenlegung des Quellcodes, die Angabe von Lizenz- und Urheberrechtshinweisen sowie

die Veröffentlichung unter identischen Lizenzbedingungen. Beim Selbst-Hosting, der gängigen Betriebsform, entstehen keine zusätzlichen Kosten. Kosten fallen lediglich an, wenn die optionale Cloud-Funktionalität genutzt wird.

Datenschutz (NFA 8)

Beim Selbst-Hosting verbleiben sämtliche Daten in der eigenen Umgebung. Für die Nutzung der optionalen Cloud-Funktion gibt es keine spezifischen Angaben zum Datenschutz.

Benutzeroberfläche (NFA 9)

Die Benutzeroberfläche von openHAB ist komplex, da sie viele Einstellungsmöglichkeiten bietet. Dies kann jedoch bei Einsteigern für Überforderung sorgen (openHAB 2025i).

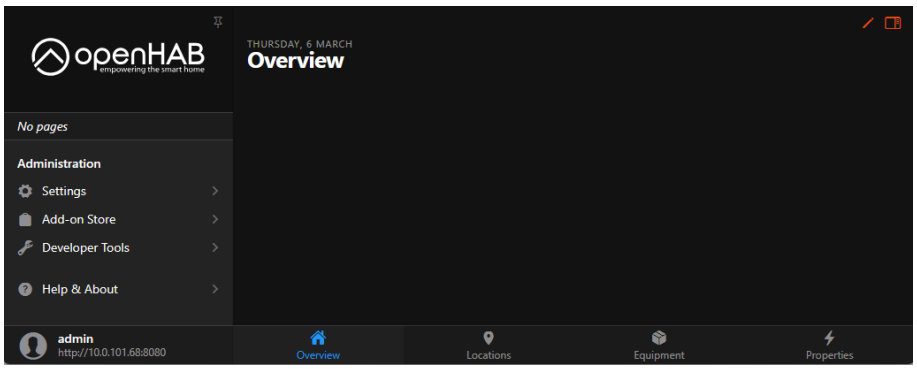

Abbildung 4.4: Standard Benutzeroberfläche openHAB als Admin.

Kompatibilität (NFA 10)

openHAB besitzt einen modularen Aufbau und bietet neben einer professionellen, gepflegten Dokumentation auch Support für über 3000 Things (Geräte) (openHAB 2025f). Dafür stehen aktuell rund 400 Bindings zur Verfügung (openHAB 2025a). Darüber hinaus existieren zusätzliche Erweiterungsmöglichkeiten wie Automation Engine Module, Transformation/Profiles, IO Services, Natural Language Processing Skills, Persistence Services sowie Audio-Sinks/-Sources (openHAB 2025b).

4.1.3 Home Assistant

Home Assistant verspricht eine leistungsstarke Smart-Home-Lösung zu sein, die besonderen Wert auf lokale Kontrolle und Datenschutz legt (Home Assistant 2025e). Es handelt sich um ein Open-Source-Projekt, das von der Open Home Foundation entwickelt und durch Nabu Casa finanziert wird (Home Assistant 2025e).

Verarbeitung der Sensordaten (FA 1)

Zur Steuerung von Abläufen können benutzerdefinierte Automatisierungsregeln definiert werden (FA 1a) (Home Assistant 2025b). Darüber hinaus besteht die Möglichkeit, eigene Integrationen in Python zu entwickeln, um die Funktionalität gezielt zu erweitern (Home Assistant 2025c). Diese ermöglichen das Erzeugen von Nachrichten (FA 1c) und deren Weiterleitung an ein Backend zur Zustellung von Push-Benachrichtigungen (FA 1b).

Skalierbarkeit (NFA 2)

Konkrete Aussagen zur maximalen Skalierbarkeit sind nicht dokumentiert. Die mögliche Anzahl integrierter Sensoren dürfte jedoch maßgeblich von den jeweils eingesetzten Integrationen abhängen.

Reaktionszeiten (NFA 3)

Home Assistant macht keine Angaben zu Reaktionszeiten, jedoch hängt dies auch von den verwendeten Integrationen ab.

Wartbarkeit und Erweiterbarkeit (NFA 4)

Da Home Assistant als Open-Source-Plattform verfügbar ist (Home Assistant 2025e), bietet es die Möglichkeit, nach Bedarf gewartet und angepasst zu werden, um verschiedenen Anforderungen gerecht zu werden. Die modular aufgebaute Architektur ermöglicht es, neue Integrationen während des laufenden Betriebs hinzuzufügen, wodurch Home Assistant eine hohe Erweiterbarkeit erreicht (Home Assistant 2025a).

Ressourcenverbrauch (NFA 5)

Es gibt keine Angaben zum Ressourcenverbrauch. Dieser hängt von der Anzahl der verwendeten Integrationen, Automationen und der Hardware-Infrastruktur ab.

Aktualität und Zukunftsfähigkeit (NFA 6)

Seit der Einführung Ende 2017 hat sich Home Assistant zu einem der erfolgreichsten Open-Source-Projekte weltweit entwickelt. Die Plattform ist besonders populär in der Community, was sich in der großen Zahl an Mitwirkenden auf GitHub widerspiegelt (GitHub Staff 2024). Es gibt ein aktives Forum sowie einen Discord-Server, die als zentrale Anlaufstellen für Support und Austausch dienen. Updates werden regelmäßig veröffentlicht, in der Regel immer am ersten Mittwoch im Monat (Home Assistant 2025d), was die kontinuierliche Weiterentwicklung der Plattform unterstreicht.

Kosten und Lizenz (NFA 7)

Home Assistant ist unter der Apache 2.0 Lizenz veröffentlicht (Home Assistant 2018). Diese Lizenz erlaubt kommerzielle Nutzung und Modifikationen, setzt jedoch voraus, dass Änderungen dokumentiert werden und ein Lizenz- sowie Urheberrechtsvermerk enthalten sind. Die gängige Betriebsform ist das Selbst-Hosting, bei dem keine zusätzlichen Kosten entstehen. Für Nutzer, die die optionale Cloud-Funktionalität nutzen möchten, fallen jedoch Gebühren an (NABU CASA 2025a). Diese kostenpflichtige Cloud-Dienstleistung, „Home Assistant Cloud“, wird von Nabu Casa angeboten und erleichtert unter anderem die Integration mit Sprachassistenten und Remote-Zugriff.

Datenschutz (NFA 8)

Beim Selbst-Hosting verbleiben sämtliche Daten in der eigenen Infrastruktur. Wird hingegen der Cloud-Dienst von Nabu Casa genutzt, erfolgt eine Datenübertragung an deren Server. Laut Angaben der Entwickler wurde Home Assistant von Beginn an mit Blick auf Datenschutz konzipiert. Durch den offenen Quellcode ist zudem transparent nachvollziehbar, welche Daten verarbeitet und gespeichert werden (NABU CASA 2025b).

Benutzeroberfläche (NFA 9)

Die Benutzeroberfläche von Home Assistant basiert auf einem TypeScript-Frontend und ist für ihre hohe Anpassbarkeit bekannt. Benutzer können eigene Dashboards erstellen und das Design an ihre individuellen Bedürfnisse anpassen. Dies bietet eine intuitive Bedienung für technisch versierte Nutzer (Home Assistant 2022). Zudem gibt es standartmäßig bereits eine vorgefertigte Oberfläche.

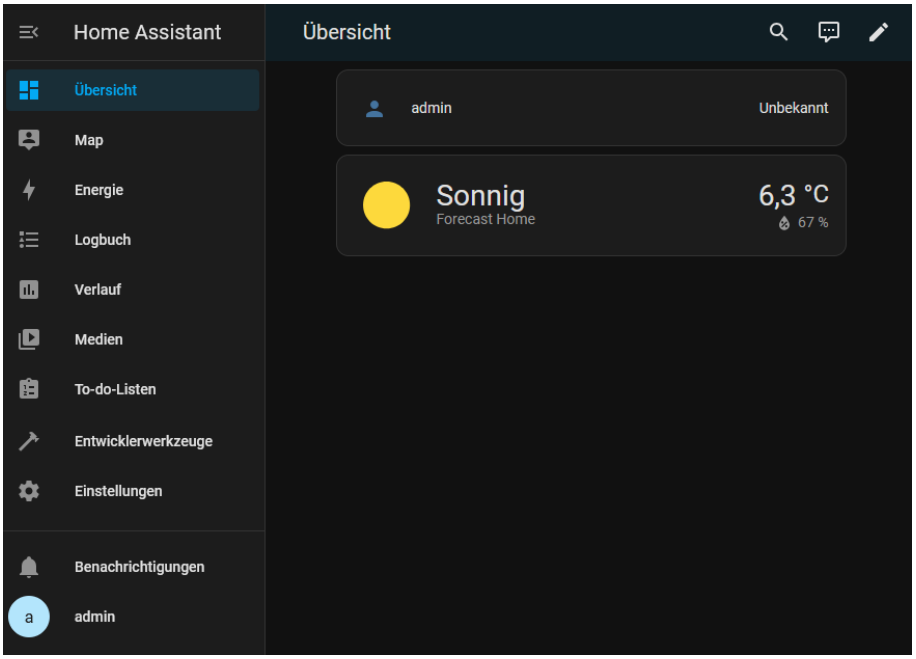

Abbildung 4.5: Standard Benutzeroberfläche Home Assistant als Admin.

Kompatibilität (NFA 10)

Home Assistant bietet eine hohe Flexibilität, da es mehr als 3 000 Integrationen (Home Assistant 2025e) unterstützt und mit über 1 000 Marken kompatibel ist (Home Assistant 2025e).

4.1.4 Gegenüberstellung und Bewertung

ioBroker, openHAB und Home Assistant bieten unterschiedliche Stärken im Bereich der Smart-Home-Automatisierung.

Die Anforderung an die Verarbeitung der Sensordaten (FA 1) ist bei allen Plattformen gegeben. Die Unterschiede liegen lediglich in der Programmiersprachen. Erweiterungen für ioBroker werden in JavaScript geschrieben, für openHAB in Java und für Home Assistant in Python.

Hinsichtlich der Skalierbarkeit (NFA 2) geben ioBroker, openHAB und Home Assistant keine konkreten Angaben. Bei allen drei Systemen könnte die Anzahl der unterstützten Sensoren von den jeweils verwendeten Adaptern, Bindings oder Integrationen abhängen, was die Skalierbarkeit je nach Konfiguration variieren lässt.

Auch die Fähigkeit zur unmittelbaren Datenverarbeitung (NFA 3) ist bei allen Plattformen vorhanden, hängt jedoch von den eingesetzten Adaptern, Bindings oder Integrationen ab.

Die Wartbarkeit und Erweiterbarkeit (NFA 4) ist bei ioBroker, openHAB und Home Assistant gleichermaßen gegeben. Neue Erweiterungen lassen sich jederzeit integrieren, und dank der Open-Source-Basis wäre auch eine eigene Wartung oder Weiterentwicklung realisierbar.

Hinsichtlich des Ressourcenverbrauchs (NFA 5) gibt es bei allen Plattformen keine konkreten Angaben. Jedoch hängt dieser auch von der Auslastung des Systems ab.

In Bezug auf Aktualität und Zukunftsfähigkeit (NFA 6) zeigen sich unterschiedliche Entwicklungen. ioBroker, das 2014 eingeführt wurde, hat weltweit an Aktivität verloren, insbesondere außerhalb des deutschsprachigen Raums. Dennoch werden etwa alle sechs Monate größere Updates veröffentlicht, um die Plattform aktuell zu halten. openHAB, das bereits 2010 auf den Markt kam, wird kontinuierlich weiterentwickelt und profitiert von einer aktiven Community, die regelmäßig Minor-Versionen bereitstellt. Home Assistant, das erst Ende 2017 eingeführt wurde, hat sich in kurzer Zeit zu einem der erfolgreichsten Open-Source-Projekte weltweit entwickelt. Die Plattform zeichnet sich durch monatliche Updates aus und ist deutlich gefragter ist als die anderen beiden Plattformen (siehe Abb. 4.6, S. 19).

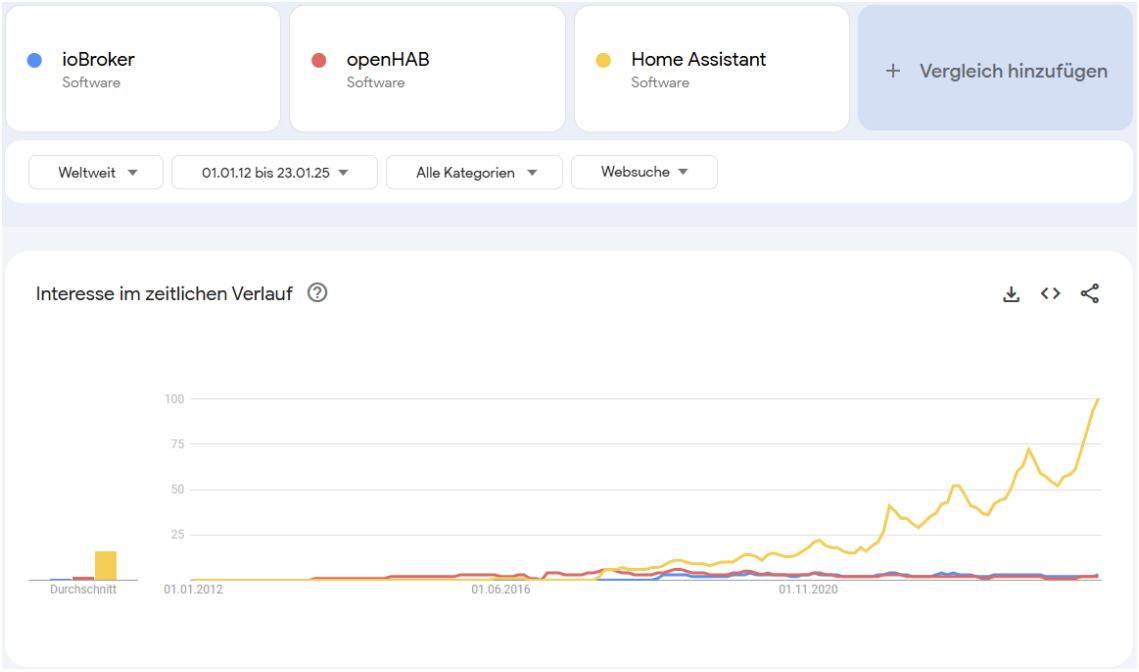

Abbildung 4.6: Suchinteresse von ioBroker, openHAB und Home Assistant. (Quelle: Google Trends 2025)

Gemeinsam ist allen Plattformen, dass beim Selbst-Hosting keine zusätzlichen Kosten entstehen, während die Nutzung von Cloud-Diensten kostenpflichtig ist (NFA 7a). Alle drei Plattformen sind Open-Source-Projekte, unterscheiden sich jedoch in ihrer Lizenzierung (NFA 7b). ioBroker ist unter der MIT-Lizenz verfügbar. openHAB hingegen nutzt die Eclipse Public License 2.0, die jedoch die Offenlegung des Quellcodes und die Bereitstellung unter derselben Lizenz vorschreibt. Home Assistant verwendet die Apache 2.0 Lizenz, die ebenfalls kommerzielle Nutzung erlaubt.

Der Datenschutz (NFA 8) ist bei allen drei Plattformen gewährleistet, wenn sie lokal bereitgestellt werden, da sämtliche Daten in der eigenen Umgebung verbleiben. Bei der Nutzung von Cloud-Diensten gibt es Unterschiede: Während ioBroker und openHAB hierzu keine konkreten Angaben machen, betont Home Assistant, dass die Plattform von Grund auf mit besonderem Fokus auf Datenschutz entwickelt wurde.

Die Benutzeroberflächen (NFA 9) der Plattformen unterscheiden sich. Dennoch bieten alle Plattformen standardisierte Benutzeroberflächen, die sich flexibel an die jeweiligen Bedürfnisse und Anforderungen anpassen lassen.

Im Bereich der Funktionalität (NFA 10) zeigt sich, dass ioBroker mit über 640 verfügbaren Adaptern eine breite Palette an Erweiterungsmöglichkeiten bietet, wodurch fast alle freien und kommerziellen Produkte integriert werden können. Auch bei openHAB stehen ähnliche Erweiterungsmöglichkeiten zur Verfügung: Mit Unterstützung für über 3000 Geräte, 400 verfügbaren Bindings. Home Assistant bietet mit über 3000 Integrationen und der Unterstützung für mehr als 1 000 Marken die umfangreichste Gerätekompatibilität.

Tabelle 4.1: Vergleich der Integrationsplattformen

Anforderung	**ioBroker**	**openHAB**	**Home Assistant**
Verarbeitung der Sensordaten (FA 1)	✓ JavaScript (TS)	✓ Java	✓ Python
Skalierbarkeit (NFA 2)	k.A.	k.A.	k.A.
Reaktionszeit (NFA 3)	✓ abhängig vom Adapter	✓ abhängig vom Binding	✓ abhängig von der Integration
Wartbarkeit und Erweiterbarkeit (NFA 4)	✓	✓	✓
Ressourcenverbrauch (NFA 5)	k.A.	k.A.	k.A.
Aktualität und Zukunftsfähigkeit (NFA 6)	seit 2014, rückläufig Major-Version alle sechs Monate	seit 2010 Minor-Version alle sechs Monate	seit 2017, Top Open-Source-Projekt Update jeden ersten Mittwoch im Monat
Kosten (NFA 7a)	Open-Source, Cloud teilweise kostenpflichtig	Open-Source, Cloud kostenpflichtig	Open-Source, Cloud kostenpflichtig
Lizenz (NFA 7b)	MIT	Eclipse Public License 2.0	Apache 2.0 License
Datenschutz (NFA 8)	✓	✓	✓
Benutzeroberfläche (NFA 9)	anpassbar mit Visualisierungsadaptern, teilweise kostenpflichtig	sehr viele Einstellungsmöglichkeiten	anpassbar
Kompatibilität (NFA 10)	> 640 Adapter	> 400 Bindings	> 3000 Integrationen

Aus der Gegenüberstellung geht hervor, dass grundsätzlich alle drei Plattformen für den Anwendungsfall eines generischen Benachrichtigungssystems geeignet sind. Die größte Verbreitung, auch in Deutschland, hat jedoch Home Assistant. Daher wäre es sicherlich sinnvoll, eine Integration für Home Assistant zu realisieren, da es sowohl für Einsteiger als auch für technikaffine Nutzer gut geeignet ist. Perspektivisch gesehen wäre es jedoch von Vorteil, eine Erweiterung für alle drei Plattformen anzubieten. So kann eine möglichst breite Nutzerbasis angesprochen und die Reichweite des Systems erhöht werden.

4.2 Analyse bestehender Systeme

Die vorliegende Analyse untersucht die vier bestehenden Push-Benachrichtigungssysteme Pushsafer, Pushover, Pushbullet und Gotify in Bezug auf die Erfüllung der definierten funktionalen und nichtfunktionalen Anforderungen des generischen Benachrichtigungssystems. Die Auswahl der Push-Benachrichtigungssysteme Pushsafer, Pushover und Pushbullet basiert auf ihrer breiten Verbreitung und ihrer Kompatibilität mit allen drei untersuchten Integrationsplattformen. Gotify wurde aufgrund seines Open-Source-Charakters und seiner Verwendung einer alternativen Push-Technologie gewählt, die einen differenzierten Ansatz im Vergleich zu den anderen Systemen bietet.

Diese Systeme stellen keine monolithische Lösung dar, sondern bieten Erweiterungen für verschiedene Integrationsplattformen an. Diese flexible Regelung und Verwaltung der Sensoren erleichtert die Abstimmung der Systeme auf spezifische Anforderungen und unterstützt die effiziente Nutzung der jeweiligen Stärken der Push-Benachrichtigungssysteme. Die Push-Benachrichtigungssysteme fungieren als Komponente zwischen der Integrationsplattform und der Individualanwendung (siehe Abb. 4.7).

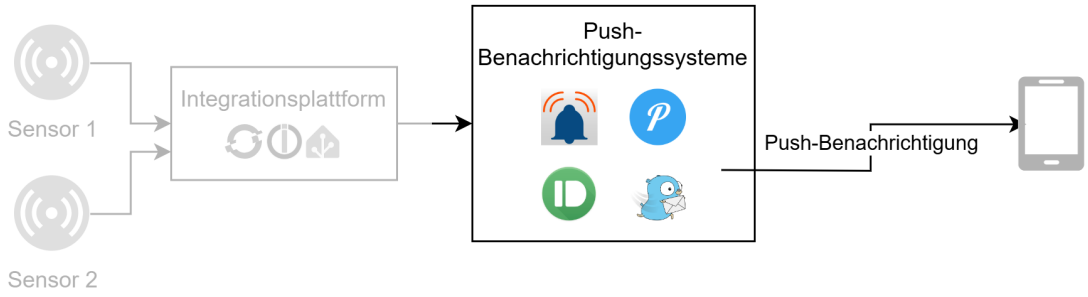

Abbildung 4.7: Systemarchitektur für ein generisches Benachrichtigungssystem mit bestehenden Lösungen.

Aus diesem Grund werden im folgenden Vergleich nicht alle funktionalen Anforderungen überprüft, da einige ausschließlich die Integrationsplattformen, den Sensor oder die App betreffen (FA 1 und FA 5a, siehe Tab A.1, S. 101). Außerdem fehlen Angaben oder Untersuchungen hinsichtlich der Zuverlässigkeit (NFA 1).

4.2.1 Pushsafer

Pushsafer ist ein in Deutschland betriebener und bereitgestellter Service zum Senden und Empfangen von Push-Benachrichtigungen. Dank zahlreicher Plugins lässt sich der Dienst einfach in bestehende Integrationsplattformen und Systeme integrieren (Pushsafer 2025e).

Funktionalitäten

Eine API ermöglicht das Versenden von Benachrichtigungen (Pushsafer 2025d), ist jedoch funktional auf die Nutzung mit den proprietären Clients des Anbieters beschränkt. Diese Einschränkung macht es unmöglich Pushsafer in externe Anwendungen oder Systeme zu integrieren (FA 5b), da eine Nutzung außerhalb der eigenen Infrastruktur nicht vorgesehen ist (Pushsafer 2025b). Die Speicherung und Abrufbarkeit von Benachrichtigungen (FA 2) erfolgt bei Pushsafer ausschließlich gerätebasiert, ohne Such- oder Filterfunktionen für einen Benutzer. Funktionen zur Nutzerverwaltung (FA 3) werden nicht angeboten, da Pushsafer darauf ausgelegt ist, die Registrierung und Verwaltung durch Benutzeraktionen über eigene Plattformen abzuwickeln. Eine Zuordnung von Nachrichten zu spezifischen Geräten (FA 4) ist jedoch möglich, was eine differenzierte Ansprache erlaubt.

Skalierbarkeit (NFA 2)

Die Anzahl versendeter Benachrichtigungen lässt sich nahezu unbegrenzt skalieren. Allerdings steigen die Kosten mit zunehmender Nutzung, da das Preismodell auf der Anzahl der gesendeten Nachrichten basiert. Konkrete Angaben zur maximal unterstützten Nutzer- und Sensorenzahl fehlen. Letztere kann zudem von der verwendeten Integrationsplattform abhängen.

Reaktionszeiten (NFA 3)

Pushsafer wirbt explizit mit einer geringen Reaktionszeit und Echtzeit-Benachrichtigungen (Pushsafer 2025e).

Wartbarkeit und Erweiterbarkeit (NFA 4)

Im Hinblick auf die Wartbarkeit und Erweiterbarkeit stellt Pushsafer 72 Plugins für diverse Plattformen bereit (Pushsafer 2025a), was eine Integration in verschiedene Systeme ermöglicht.

Ressourcennutzung (NFA 5)

Die Ressourcennutzung bei Pushsafer stellt keine wesentliche Herausforderung dar, da der Dienst cloudbasiert ist und somit keine Belastung für lokale Systeme verursacht.

Aktualität und Zukunftsfähigkeit (NFA 6)

Die aktive Weiterentwicklung des Dienstes zeigt sich in regelmäßigen Software-Updates und einem lebhaften Forum mit administrativer Beteiligung (Pushsafer 2025i). Diese kontinuierliche Entwicklung spricht für die langfristige Aktualität und Zukunftsfähigkeit des Dienstes. Im Google Play Store erhält Pushsafer eine Nutzerbewertung von 4,5 Sternen (Google Play 2025d), was auf eine insgesamt positive Nutzererfahrung hinweist. Mit mehr als 10 000 Downloads der App (Google Play 2025d) zeigt der Dienst eine solide Nutzerbasis.

Kosten und Lizenz (NFA 7)

Das flexible Preismodell von Pushsafer reicht von 0,99€ für 1 000 API-Aufrufe bis zu 99,99€ für 500 000 API-Aufrufe (Pushsafer 2025c). Angaben zur kommerziellen Nutzung sind nicht verfügbar.

Datenschutz (NFA 8)

Pushsafer gewährleistet einen hohen Datenschutzstandard durch seinen Hosting-Standort in Deutschland.

Benutzerfreundlichkeit (NFA 9)

Durch eine benutzerfreundliche und intuitive Oberfläche ermöglicht Pushsafer eine einfache Nutzung des Dienstes. Die Gestaltung der Benutzeroberfläche ist ansprechend und fördert eine schnelle Eingewöhnung der Nutzer. Durch die Anmeldung auf der Website bekommen die Nutzer direkt Zugriff auf die grafische Oberfläche.

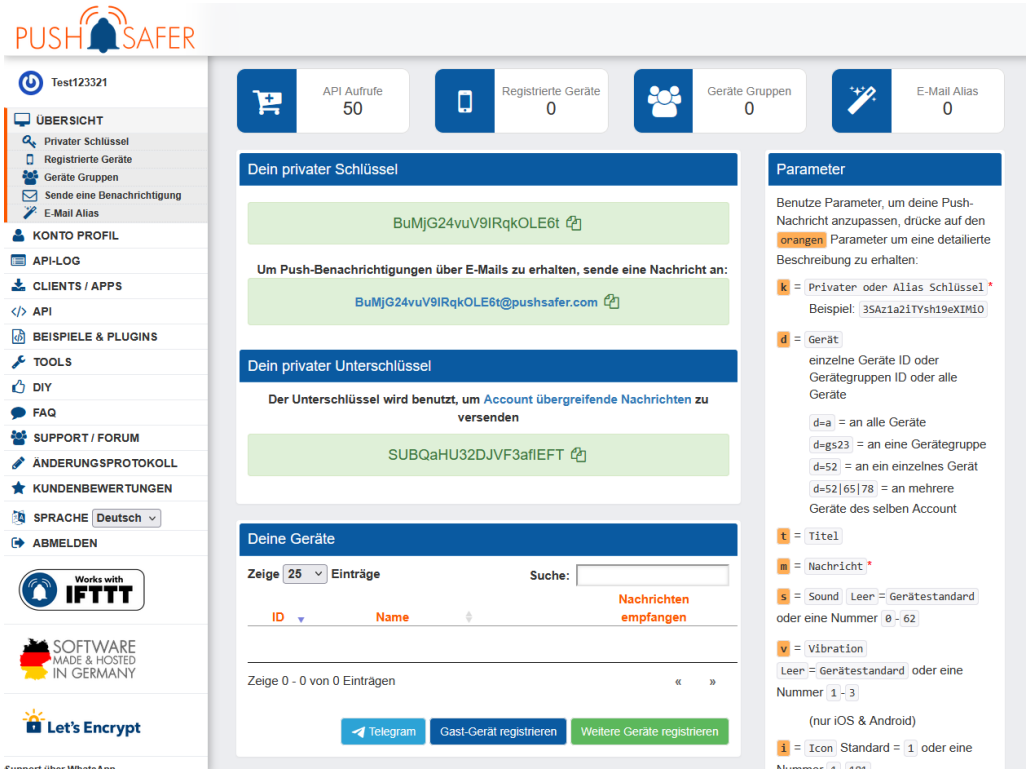

Abbildung 4.8: Dashboard Pushsafer. (Quelle: Pushsafer 2025f)

Kompatibilität (NFA 10)

Die Verfügbarkeit von Erweiterungen für gängige Integrationsplattformen wie ioBroker (Fischer 2024), openHAB (Pushsafer 2025h) und Home Assistant (Pushsafer 2025g) gewährleistet eine breite Kompatibilität mit etablierten Protokollen und eine flexible Einbindung in bestehende Systeme.

Portabilität (NFA 11)

Pushsafer erfüllt die definierten Anforderungen an die Portabilität, indem es die Plattformen iOS, Android und Web unterstützt (Pushsafer 2025e). Diese plattformübergreifende Verfügbarkeit ermöglicht eine flexible Nutzung und erleichtert die Integration in bestehende Systemlandschaften.

4.2.2 Pushover

Pushover verspricht eine einfache Möglichkeit, Echtzeit-Benachrichtigungen direkt auf verschiedenen Geräten zu empfangen (Pushover 2024).

Funktionalitäten

Zur Anbindung an individuelle Anwendungen (FA 5b) bietet der Dienst sowohl eine Message API (Pushover 2025b) als auch eine offene Client API (Pushover 2025e) an. Diese Schnittstellen ermöglichen die nahtlose Integration von Pushover in individuelle Anwendungen, indem sie eine Übermittlung von Benachrichtigungen sowie den Empfang und die Verwaltung eingehender Nachrichten unterstützen. Pushover verzichtet vollständig auf die Speicherung und Abrufbarkeit von bereits zugestellten Benachrichtigungen (FA 2) und bietet keine explizite API-Unterstützung für eine umfassende Nutzerverwaltung (FA 3), da weder entsprechende API-Endpunkte bereitgestellt werden noch die Registrierung der Nutzer außerhalb der eigenen Plattform möglich ist. Die Geräteverwaltung (FA 4) wird zwar unterstützt, ist jedoch stark an die proprietären Clients des Systems gebunden.

Skalierbarkeit (NFA 2)

Die Plattform ermöglicht eine hohe Skalierbarkeit der Benachrichtigungen, wobei eine zunehmende Nutzung mit höheren Kosten einhergeht (Pushover-Support 2021). Informationen zur maximalen Anzahl an Nutzern und Sensoren sind nicht verfügbar.

Reaktionszeiten (NFA 3)

Pushover hebt ausdrücklich seine geringe Reaktionszeit und die Bereitstellung von Echtzeit-Benachrichtigungen hervor (Pushover 2024).

Wartbarkeit und Erweiterbarkeit (NFA 4)

Eine breite Unterstützung von 87 offiziell aufgelisteten Anwendungen, Plugins und Websites, die mit dem Pushover-Dienst verwenden können (Pushover 2025g) erleichtern Erweiterung des Systems.

Ressourcennutzung (NFA 5)

Die cloudbasierte Architektur minimiert die Belastung lokaler Systeme, da keine signifikanten Ressourcen auf den Endgeräten beansprucht werden.

Aktualität und Zukunftsfähigkeit (NFA 6)

Regelmäßige Updates aller sechs Monate (Stein 2024) und kontinuierliche Pflege zeugen von einer nachhaltigen Weiterentwicklung. Mit einer Bewertung von 4,6 Sternen im Google Play Store (Google Play 2025c) zeigt sich eine hohe Nutzerzufriedenheit. Pushover hat mehr als 100 000 Downloads im Play Store (Google Play 2025c) und bietet damit eine breite Nutzerbasis.

Kosten und Lizenz (NFA 7)

Pushover erlaubt eine kostenfreie Nutzung von bis zu 10 000 Nachrichten pro Monat. Nutzer, die zusätzliche Nachrichtenkontingente benötigen, können diese hinzubuchen. Die Preise für zusätzliche Kapazitäten reichen von $50 USD für 10 000 zusätzliche Nachrichten bis zu $1000 USD für 500 000 Nachrichten (Pushover-Support 2021). Es gibt jedoch keine spezifischen Informationen zur kommerziellen Nutzung. Jeder Nutzer, der eine Anwendung basierend auf der Pushover Open Client API verwendet, muss eine Pushover für Desktop-Lizenz erwerben (Pushover 2025e). Diese Lizenz kostet einmalig $4,99 USD pro Nutzer (Pushover 2025d).

Datenschutz (NFA 8)

Es existieren keine weiterführenden Informationen zum Datenschutz. Die Server, an die API-Aufrufe gerichtet werden, befinden sich in den USA (Pushover 2025f).

Benutzerfreundlichkeit (NFA 9)

Pushover zeichnet sich durch eine einfache und übersichtliche Benutzeroberfläche aus. Die Nutzer können problemlos Benachrichtigungen und Geräte verwalten und anpassen, ohne dass umfangreiche technische Kenntnisse erforderlich sind.

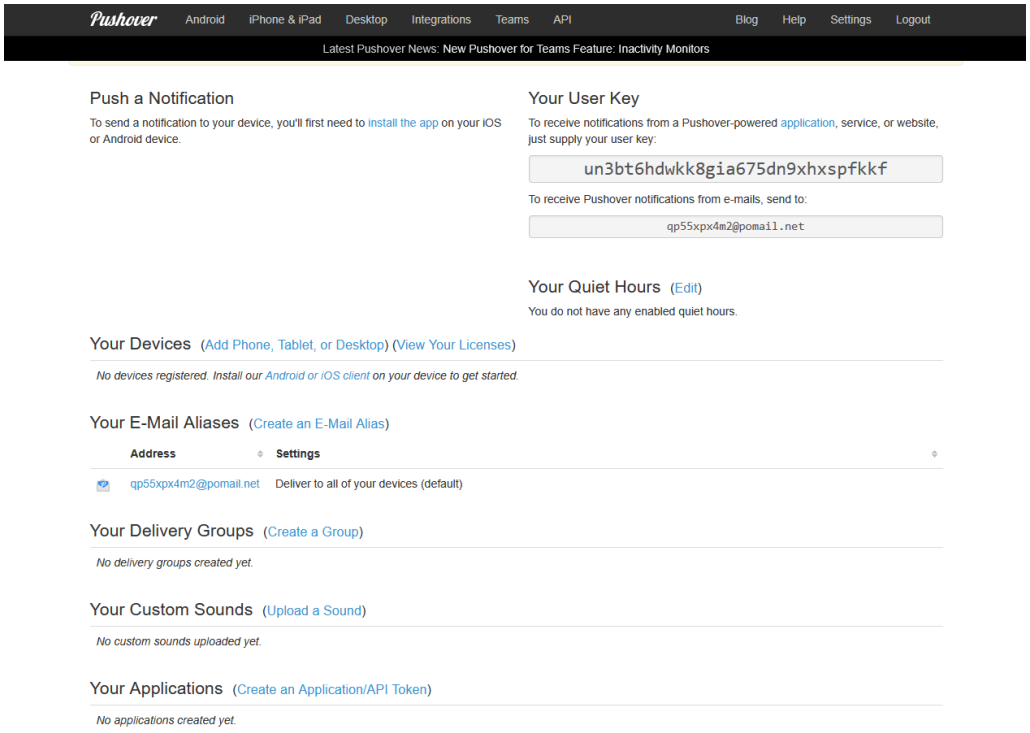

Abbildung 4.9: Benutzeroberfläche Pushover. (Quelle: Pushover 2024)

Kompatibilität (NFA 10)

Durch die Bereitstellung von Erweiterungen für etablierte Integrationsplattformen wie ioBroker (Haev 2024a), openHAB (Pushover 2025c) und Home Assistant (Pushover 2025a) weist Pushover eine hohe Kompatibilität auf.

Portabilität (NFA 11)

Die plattformunabhängige Nutzung wird durch die Unterstützung von iOS und Android (Pushover 2024), jedoch fehlt die Unterstützung für Web-Pushes.

4.2.3 Pushbullet

Pushbullet vernetzt Geräte nahtlos miteinander, sodass sie sich wie ein einheitliches System anfühlen (Pushbullet 2020b). Zusätzlich bietet der Dienst Plugins für zahlreiche Integrationsplattformen, um die Integration von Sensoren oder bestehenden Systemen zu erleichtern.

Funktionalitäten

Eine umfassende API (Pushbullet 2020c) ermöglicht die Integration in eigene Anwendungen (FA 5b). Allerdings beschränkt sich der Funktionsumfang der API

auf das Senden von Push-Benachrichtigungen, während der Empfang von Benachrichtigungen durch eigene Anwendungen nicht unterstützt wird. Benachrichtigungen können gespeichert und abgerufen werden (FA 2). Aufgrund fehlender Mandantenfähigkeit (FA 3) verzichtet Pushbullet auf eine Nutzerverwaltung und eine Filterung für spezifische Nutzer. Die Zuordnung von Nachrichten zu spezifischen Geräten (FA 4) wird unterstützt, was eine zielgerichtete Kommunikation ermöglicht.

Skalierbarkeit (NFA 2)

In der kostenfreien Version ist die Anzahl der Push-Nachrichten auf maximal 500 pro Monat begrenzt (Pushbullet 2020a), was insbesondere für Anwendungen mit hohem Nachrichtenaufkommen eine erhebliche Einschränkung darstellt. Eine größere Skalierbarkeit erfordert den Wechsel zu einem kostenpflichtigen Tarif. Da das System keine Mandantenverwaltung unterstützt, ist die Nutzeranzahl vollständig begrenzt. Hinsichtlich der maximal unterstützten Sensorenzahl liegen keine Informationen vor.

Reaktionszeiten (NFA 3)

Für Pushbullet gibt es keine offiziellen Angaben zur Latenz.

Wartbarkeit und Erweiterbarkeit (NFA 4)

Eine offizielle Liste unterstützter Plugins existiert nicht. Dennoch ist der Dienst mit weit verbreiteten Smart-Home-Plattformen wie ioBroker, Home Assistant und openHAB kompatibel, was die Integration erleichtert.

Ressourcennutzung (NFA 5)

Die Ressourcennutzung von Pushbullet ist irrelevant, da der Dienst in der Cloud bereitgestellt wird und keine Belastung für lokale Systeme darstellt.

Aktualität und Zukunftsfähigkeit (NFA 6)

Mehr als fünf Millionen Downloads im Google Play Store (Google Play 2025b) unterstreichen die Popularität. Trotz der weiten Verbreitung bleibt die Zukunftsfähigkeit unklar, da keine offiziellen Angaben zur Weiterentwicklung vorliegen.

Kosten und Lizenz (NFA 7)

Die kostenfreie Nutzung von Pushbullet ist auf 500 Push-Nachrichten pro Monat limitiert (Pushbullet 2020a). Konkrete Angaben zu den Erweiterungsmöglichkeiten der kostenpflichtigen Version fehlen, ebenso wie Informationen zur rechtlichen Zulässigkeit einer kommerziellen Nutzung.

Datenschutz (NFA 8)

Zwar wird eine Ende-zu-Ende-Verschlüsselung verwendet, jedoch bleibt unklar, ob der Anbieter theoretisch Zugriff auf die übertragenen Inhalte erlangen könnte (Langer 2015).

Benutzerfreundlichkeit (NFA 9)

Die Benutzeroberfläche ist übersichtlich und leicht verständlich gestaltet. Beim ersten Aufruf der Anwendung im Web wird der Nutzer direkt auf eine Setup-Seite geleitet, die eine verständliche und schrittweise Anleitung zur Einrichtung der Anwendung bietet.

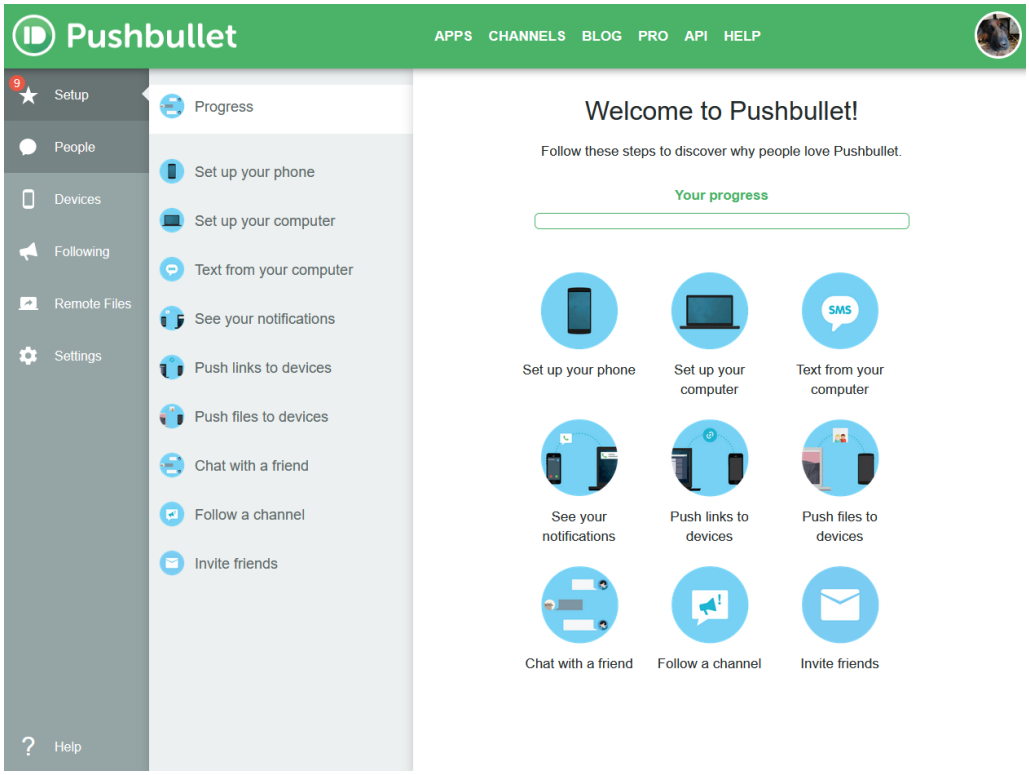

Abbildung 4.10: Setup Pushbullet. (Quelle: Pushbullet 2025c)

Kompatibilität (NFA 10)

Pushbullet zeichnet sich durch eine hohe Kompatibilität aus, da Erweiterungen für gängige Integrationsplattformen wie ioBroker (M. 2024), openHAB (Pushbullet 2025b) und Home Assistant (Pushbullet 2025a) bereitstellt werden.

Portabilität (NFA 11)

Der Empfang von Benachrichtigungen ist auf Android und im Web möglich (Pushbullet 2020b), während eine Unterstützung für iOS fehlt.

4.2.4 Gotify

Gotify ist ein einfacher Server zur Übermittlung und zum Empfang von Nachrichten (Gotify 2025a).

Funktionalitäten

Eine umfassende API (Gotify 2025b) ermöglicht die Integration in eigene Anwendungen (FA 5b). Der Nachrichtenaustausch erfolgt bei Android Geräten über WebSocket-Verbindungen (Gotify 2025a). Allerdings führt eine dauerhafte Websocket Verbindung zu einem hohen Stromverbrauch, was auf mobilen Endgeräten ein Problem darstellen könnte. Für iOS gibt es eine inoffizielle Unterstützung, die den Apple Push Notification Service (= APNs) verwendet (Sebastian 2025). Als Open-Source-Lösung ermöglicht Gotify jedoch die Anpassung und Erweiterung der Software, sodass sie flexibel an individuelle Anforderungen angepasst werden kann. Eine integrierte Funktion zur Speicherung und zum Abruf von Benachrichtigungen ist nicht vorhanden (FA 2). Dennoch stehen API-Funktionen bereit, die eine umfassende Nutzerverwaltung (FA 3) ermöglichen, einschließlich der Erstellung, Aktualisierung und Verwaltung von Benutzern. Eine Geräteverwaltung wird hingegen nicht unterstützt (FA 4).

Skalierbarkeit (NFA 2)

Gotify, als Open-Source-Software, bietet eine unbegrenzte Skalierbarkeit, da keine festen Nutzungsgrenzen bestehen. Allerdings erfordert der Betrieb eigene Hosting-Ressourcen, deren Kosten mit zunehmender Skalierung steigen können. Zudem ist nicht eindeutig dokumentiert, wie effizient der Code geschrieben ist, sodass bei hoher Auslastung potenziell Verzögerungen oder Stabilitätsprobleme auftreten könnten.

Reaktionszeiten (NFA 3)

Echtzeit-Benachrichtigungen werden nach eigenen Angaben von Gotify unterstützt (Gotify 2025c), wobei die tatsächliche Latenz von der individuellen Serverimplementierung und den verfügbaren Ressourcen abhängt.

Wartbarkeit und Erweiterbarkeit (NFA 4)

Eine offizielle Liste unterstützter Plugins liegt nicht vor. Dennoch besteht Kompatibilität mit weit verbreiteten Smart-Home-Plattformen wie ioBroker, Home Assistant und openHAB, wodurch eine einfache Integration und Erweiterbarkeit durch Sensoren gewährleistet ist.

Ressourcennutzung (NFA 5)

Als selbstgehostete Lösung erfordert der Dienst eine individuelle Bereitstellung von Rechenkapazitäten. Die tatsächliche Ressourcennutzung variiert je nach Serverkonfiguration und Systemanforderungen.

Aktualität und Zukunftsfähigkeit (NFA 6)

Gotify wird kontinuierlich gepflegt und aktualisiert. Mit über 10 000 Downloads im Google Play Store (Google Play 2025a) und einer Bewertung der App von 4,9 Sternen (Google Play 2025a) zeigt Gotify eine hohe Nutzerzufriedenheit. Diese positive Resonanz sowie die kontinuierliche Weiterentwicklung lassen auf eine gute Zukunftsfähigkeit des Dienstes schließen. Gotify liegt in der Verbreitung gleichauf mit Pushsafer, beide Dienste haben mehr als 10 000 Downloads.

Kosten und Lizenz (NFA 7)

Die unter der MIT-Lizenz (Mattheis 2025) veröffentlichte Software kann uneingeschränkt angepasst und genutzt werden. Während die Nutzung kostenfrei ist, entstehen potenzielle Kosten für das Hosting und die Bereitstellung der notwendigen Infrastruktur.

Datenschutz (NFA 8)

Durch die Möglichkeit des Selbsthosting bleibt die Datensicherheit vollständig in der Hand des Nutzers. Da sämtliche Daten auf der eigenen Infrastruktur gespeichert werden können, entfällt eine Abhängigkeit von Drittanbietern.

Benutzerfreundlichkeit (NFA 9)

Eine übersichtliche Benutzeroberfläche erleichtert die Bedienung. Allerdings setzt der Betrieb die eigenständige Bereitstellung und Wartung der Hosting-Umgebung voraus, was insbesondere für technisch weniger versierte Nutzer eine Herausforderung darstellen kann.

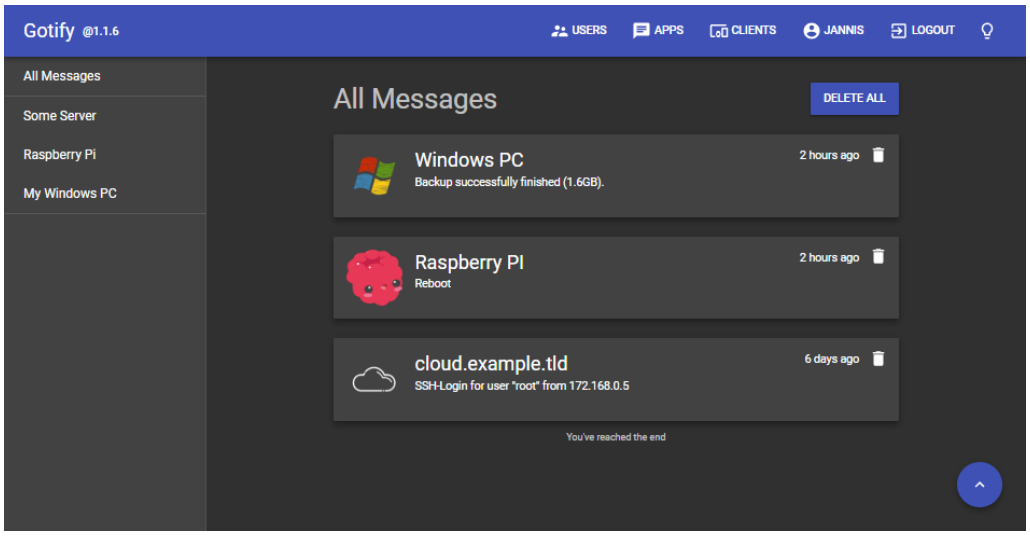

Abbildung 4.11: Benutzeroberfläche Gotify. (Quelle: Gotify 2025d)

Kompatibilität (NFA 10)

Eine solide Kompatibilität ergibt sich durch die Möglichkeit, die Integrationsplattform ioBroker über entsprechende Erweiterungen einzubinden (Pohl 2024). Darüber hinaus ermöglicht ein weiterer WebSocket-Adapter von Gotify die Anbindung eigener Anwendungen (simatec 2025).

Portabilität (NFA 11)

Die Zustellung von Push-Benachrichtigungen wird sowohl auf iOS (Sebastian 2025) als auch auf Android (Gotify 2025a) unterstützt. Jedoch werden keine Web-Pushes unterstützt.

4.2.5 Gegenüberstellung und Bewertung

Die Auswahl eines geeigneten Push-Nachrichtendienstes hängt von verschiedenen funktionalen und nichtfunktionalen Anforderungen ab. In diesem Abschnitt werden die vier untersuchten Systeme Pushsafer, Pushover, Pushbullet und Gotify systematisch gegenübergestellt und hinsichtlich ihrer Leistungsfähigkeit bewertet.

Die Integration in Anwendungen (FA 5b) wird von den vier Systemen unterschiedlich umgesetzt: Während Pushsafer stark an eigene Clients gebunden ist, stellt Pushover offene APIs zur Verfügung. Pushbullet konzentriert sich auf reine Versandfunktionen, während Gotify WebSockets nutzt. Allerdings kommt es hier vereinzelt zu Verbindungsabbrüchen. Bei der Speicherung von Benachrichtigungen (FA 2) zeigen sich ebenfalls Unterschiede. Pushsafer und Pushbullet bieten diese Funktion, wohingegen Pushover und Gotify keine Speicherung vorsehen. In Bezug auf die Nutzerverwaltung (FA 3) stellt ausschließlich Gotify umfangreiche API-Funktionen bereit. Bei der Geräteverwaltung (FA 4) erlaubt Pushsafer eine gezielte Zuordnung zu einzelnen Nutzern. Pushbullet verfügt zwar über eine Geräteverwaltung, jedoch ohne Filtermöglichkeiten. Pushover ist auf eigene Clients angewiesen, und Gotify verzichtet vollständig auf eine entsprechende Funktion.

Hinsichtlich der Skalierbarkeit (NFA 2) bieten Pushsafer und Pushover eine nahezu unbegrenzte Anzahl an Benachrichtigungen, sind jedoch kostenabhängig. Angaben zu Nutzer- oder Gerätelimits machen beide Systeme nicht. Pushbullet ist durch ein striktes Limit in der kostenlosen Version stark eingeschränkt und nicht mandantenfähig. Gotify erlaubt als Open-Source-Lösung theoretisch unbegrenzte Skalierung, erfordert aber eigene Hosting-Ressourcen und könnte bei hoher Last an Effizienzgrenzen stoßen.

Die Reaktionszeiten (NFA 3) sind bei Pushsafer und Pushover laut eigenen Angaben in Echtzeit, während Pushbullet keine Angaben zur Latenz macht. Gotify unterstützt Echtzeit-Benachrichtigungen, wobei die Geschwindigkeit von der Serverimplementierung abhängt.

Eine offizielle Liste verfügbarer Erweiterungen (NFA 4) gewährleistet bei Pushsafer und Pushover eine strukturierte und nachvollziehbare Erweiterbarkeit der Systeme. Pushbullet und Gotify bieten keine offiziellen Plugin-Listen, sind aber mit gängigen Smart-Home-Plattformen kompatibel. Die Wartbarkeit liegt außer bei Gotify komplett in der Hand der Anbieter.

Die Ressourcennutzung (NFA 5) ist bei den cloudbasierten Diensten Pushsafer, Pushover und Pushbullet unproblematisch, während Gotify als selbstgehostete Lösung von der Serverkonfiguration abhängt.

Hinsichtlich der Aktualität und Zukunftsfähigkeit (NFA 6) zeigt sich, dass Pushsafer und Pushover regelmäßig aktualisiert werden. Pushsafer und Gotify haben über 10 000 Downloads. Pushover hat mit mehr als 100 000 Downloads eine größere Nutzerbasis. Pushbullet hat über fünf Millionen Downloads, aber keine Angaben zur Weiterentwicklung. Gotify erzielt mit 4,9 Sternen im Google Play Store die höchste Zufriedenheit.

Die Kosten und Lizenzen (NFA 7) unterscheiden sich: Pushsafer hat ein gestaffeltes Preismodell, Pushover bietet eine kostenlose Nutzung bis zu 10 000 Nachrichten pro Monat, Pushbullet ist in der kostenfreien Version auf 500 Nachrichten monatlich begrenzt, und Gotify ist als Open-Source-Software kostenlos, erfordert jedoch eigene Hosting-Infrastruktur.

Der Datenschutz (NFA 8) variiert je nach System. Pushsafer profitiert von einem hohen Datenschutzniveau durch seinen Hosting Standort in Deutschland, während Pushover seine Server in den USA betreibt. Pushbullet verwendet eine Ende-zu-Ende-Verschlüsselung, aber es ist unklar, ob der Anbieter Zugriff auf die Daten hat. Gotify bietet durch Selbsthosting vollständige Kontrolle über die gespeicherten Daten und den höchsten Datenschutz.

Die Benutzerfreundlichkeit (NFA 9) der Systeme ist unterschiedlich. Alle Systeme bieten benutzerfreundliche Oberflächen, wobei Pushover eine funktionale, weniger grafisch ansprechende Benutzeroberfläche hat, was die Funktionalität jedoch nicht beeinträchtigt. Gotify erfordert ein ausgeprägteres technisches Wissen aufgrund des eigenen Hostings.

Pushsafer, Pushover und Pushbullet zeichnen sich durch eine hohe Kompatibilität (NFA 10) aus, da für alle drei Systeme Integrationen in gängige Plattformen wie ioBroker, openHAB und Home Assistant verfügbar sind. Für Gotify hingegen existiert derzeit lediglich eine Erweiterung für ioBroker, wodurch die Einbindungsmöglichkeiten eingeschränkt sind.

Die Anforderung an die Portabilität (NFA 11) wird ausschließlich von Pushsafer vollständig erfüllt. Pushover und Gotify verfügen über keine Unterstützung für Web-Push-Benachrichtigungen. Pushbullet bietet nur die Unterstützung für Android und Web, was die Nutzung einschränkt.

Tabelle 4.2: Vergleich der bestehenden Benachrichtigungssysteme

Anforderung	**Pushsafer**	**Pushover**	**Pushbullet**	**Gotify**
API/ SDK (FA 5b)	× nicht für individuelle Apps	✓	(✓) nur Versenden der Nachricht	✓
Benachrichtigungsverlauf (FA 2)	×	×	✓	×
Nutzerverwaltung (FA 3)	×	×	× nicht mandantenfähig	✓
Geräteverwaltung (FA 4)	✓	×	✓	×
Skalierbarkeit (NFA 2)	(✓) Kosten steigen	(✓) Kosten steigen	×	✓
Reaktionszeit (NFA 3)	Echtzeit	Echtzeit	k.A.	Echtzeit
Wartbarkeit und Erweiterbarkeit (NFA 4)	72 Plugins	87 Plugins/ Apps	Keine Liste, aber ioBroker, HA, OpenHAB	Keine Liste, aber ioBroker, HA, OpenHAB
Ressourcennutzung (NFA 5)	✓	✓	✓	Abhängig von eigener Infrastruktur
Aktualität und Zukunftsfähigkeit (NFA 6)	Regelmäßige Updates	Regelmäßige Updates	k.A.	Regelmäßige Updates
Kosten (monatlich) (NFA 7a)	1 000 API Aufrufe 0,99€ bis 500 000 Aufrufe 99,99€	10 000 Nachrichten gratis, mehr kostenpflichtig möglich	Max 500 Pushes/Monat gratis	Selbsthosting
Lizenz (NFA 7b)	k.A.	k.A.	k.A.	MIT
Datenschutz (NFA 8)	Standort: Deutschland	Standort: USA	Ende-zu-Ende-Verschlüsselung	Selbsthosting
Benutzerfreundlichkeit (NFA 9)	✓	✓	✓	✓

Anforderung	Pushsafer	Pushover	Pushbullet	Gotify
Kompatibilität (NFA 10)	ioBroker, openHAB, Home Assistant	ioBroker, openHAB, Home Assistant	ioBroker, openHAB, Home Assistant	ioBroker
Portabilität Android, iOS, Web (NFA 11)	✓, ✓, ✓	✓, ✓, ×	✓, ×, ✓	✓, ✓, ×

Zusammenfassend lässt sich festhalten, dass keines der untersuchten Systeme alle Anforderungen vollständig erfüllt. Pushsafer ist nicht integrierbar in eigenen Anwendungen. Pushover erfüllt nicht die Anforderungen an Speicherung und Mandantenfähigkeit. Pushbullet ist nicht mandantenfähig und scheidet somit als Lösung auch aus. Gotify hingegen hebt sich durch seine hohe Anpassungsfähigkeit und Kosteneffizienz hervor, setzt jedoch fundiertes technisches Wissen in Go voraus, insbesondere für das Hosting und die Implementierung von Erweiterungen.

Da alle untersuchten Systeme in mindestens einem Punkt den Anforderungen nicht entsprechen ergibt sich das Ziel, in den folgenden Abschnitten einzelne Komponenten miteinander zu vergleichen, um ein neues, generisches Benachrichtigungssystem zu entwickeln.

4.3 Vergleich von Push-Diensten

Push-Dienste sind ein zentrales Element moderner Anwendungen. Sie ermöglichen es Entwicklern, Nachrichten vom Backend zuverlässig an mobile Geräte und andere Plattformen zu übermitteln und als Push-Benachrichtigung anzuzeigen. Zu den bekanntesten Diensten gehören Azure Notification Hubs (= ANH), Firebase Cloud Messaging (= FCM) und OneSignal, die jeweils verschiedene Funktionen und Merkmale bieten. Um eine fundierte Entscheidung zu treffen, ist eine systematische Analyse dieser Dienste notwendig.

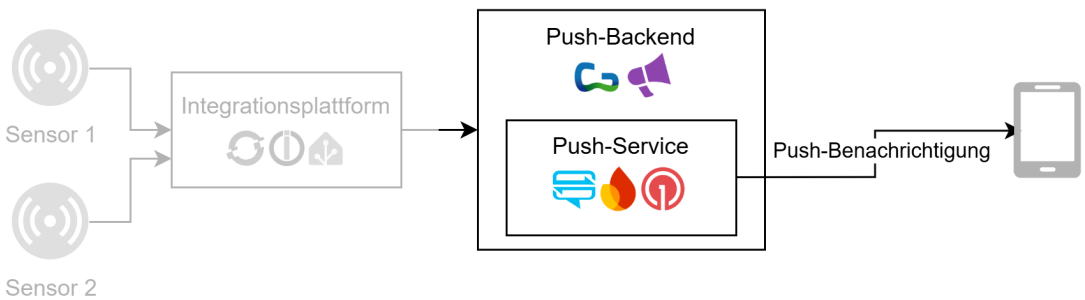

Abbildung 4.12: Push-Dienste in dem generischen Benachrichtigungssystem.

Grundlage der Bewertung bilden die zuvor definierten Anforderungen. Da fast alle funktionalen Anforderungen keinen direkten Bezug zur Push-Komponente haben, werden nur FA 1b und FA 5 berücksichtigt. Außerdem finden die Kriterien Benutzerfreundlichkeit (NFA 9) und Kompatibilität (NFA 10) keine Anwendung, da sie sich auf andere Bestandteile des Benachrichtigungssystems beziehen (siehe Tab A.1, S. 101). Ebenso ist die Ressourcennutzung (NFA 5) nicht relevant, da die drei Dienste cloudbasiert sind. Da es sich bei allen drei Diensten um proprietäre Lösungen handelt, ist eine eigenständige Wartung nicht möglich (NFA 4a), sodass Anwender vollständig von der Verfügbarkeit und Pflege durch den jeweiligen Anbieter abhängig sind.

4.3.1 Azure Notification Hubs

ANH stellt einen leistungsstarken Push-Dienst von Microsoft dar, der es Entwicklern ermöglicht, Benachrichtigungen über verschiedene Plattformen hinweg zu versenden.

Verarbeitung der Daten (FA 1b)

Die Verarbeitung von Benachrichtigungen wird unterstützt, wobei der Versand von Push-Benachrichtigungen an eine große Anzahl von Endgeräten ermöglicht wird (Microsoft 2025c).

Integration in Anwendungen (FA 5)

Mit ANH können Nachrichten auch dann an Endgeräte übermittelt werden, wenn die zugehörige Anwendung auf dem jeweiligen Gerät nicht aktiv ist. Zudem wird eine benutzerfreundliche Integration durch eine Reihe von SDKs und API-Tools ermöglicht (Microsoft 2020). Entwickler können die APIs von Azure nutzen, um Push-Benachrichtigungen problemlos in ihre Anwendungen zu integrieren.

Zuverlässigkeit (NFA 1)

Basierend auf der globalen Cloud-Infrastruktur von Microsoft Azure bietet ANH eine äußerst zuverlässige Plattform. Eine Schlüsselkomponente zur Sicherstellung der Zuverlässigkeit sind die sogenannten Verfügbarkeitszonen (= Availability Zones), die physisch getrennte Datacenter innerhalb jeder Azure-Region darstellen. Im Falle eines Ausfalls einer Zone können die Dienste automatisch auf eine andere verfügbare Zone umschalten (Microsoft 2024a). Diese Architektur stellt sicher, dass Push-Benachrichtigungen selbst bei Ausfällen einzelner Zonen weiterhin zugestellt werden.

Skalierbarkeit (NFA 2)

ANH ist auf extreme Skalierbarkeit ausgelegt und ermöglicht die Verarbeitung einer sehr großen Anzahl von Benachrichtigungen. Entwickler können Push-Benachrichtigungen an Millionen von Geräten senden, ohne eine neue Architektur einführen

oder eine horizontale Partitionierung (Sharding) vornehmen zu müssen (Microsoft 2025c). Die Direct-Send-API, die speziell für die individuelle Zustellung von Benachrichtigungen an einzelne Endnutzer konzipiert ist, ermöglicht eine maximale Versandrate von 100 000 Nachrichten pro Minute im Basic- und bis zu 1 000 000 im Standard-Tarif (Agrawal 2022). In typischen Anwendungsszenarien erfolgt die Zustellung bei 99 % der Nachrichten innerhalb von weniger als einer Sekunde (Agrawal 2022).

Reaktionszeiten (NFA 3)

Da Push-Benachrichtigungen von externen, plattformspezifischen Push Notification Services zugestellt werden, kann keine garantierte Latenzzeit angeboten werden. Die Mehrheit der Benachrichtigungen wird jedoch in der Regel innerhalb weniger Minuten zugestellt (Microsoft 2025e).

Erweiterbarkeit (NFA 4)

Die Architektur ermöglicht es der Plattform ohne notwendige Änderungen weiter zu wachsen (Microsoft 2025c).

Aktualität und Zukunftsfähigkeit (NFA 6)

ANH wird regelmäßig aktualisiert und verbessert, da es Teil des Microsoft Azure-Ökosystems ist. Obwohl keine spezifischen Details zu geplanten Änderungen vorliegen, kann davon ausgegangen werden, dass der Dienst kontinuierlich gewartet und auf dem neuesten Stand gehalten wird, um mit den technologischen Entwicklungen und Anforderungen Schritt zu halten.

Externe Dienste (NFA 7)

Die Kostenstruktur von ANH variiert in Abhängigkeit von den spezifischen Anforderungen eines Unternehmens, insbesondere der Anzahl der versendeten Benachrichtigungen sowie den genutzten Funktionen (Microsoft 2025d, siehe Abb. 4.13, S. 37), jedoch bezahlt man nur, was tatsächlich verbraucht haben. Hinsichtlich der Lizenzierung gibt es keine spezifischen Angaben, da der Dienst Bestandteil des Microsoft Azure-Cloud-Angebots ist. Die geltenden Lizenzbedingungen sind in den allgemeinen Nutzungsbedingungen von Azure enthalten.

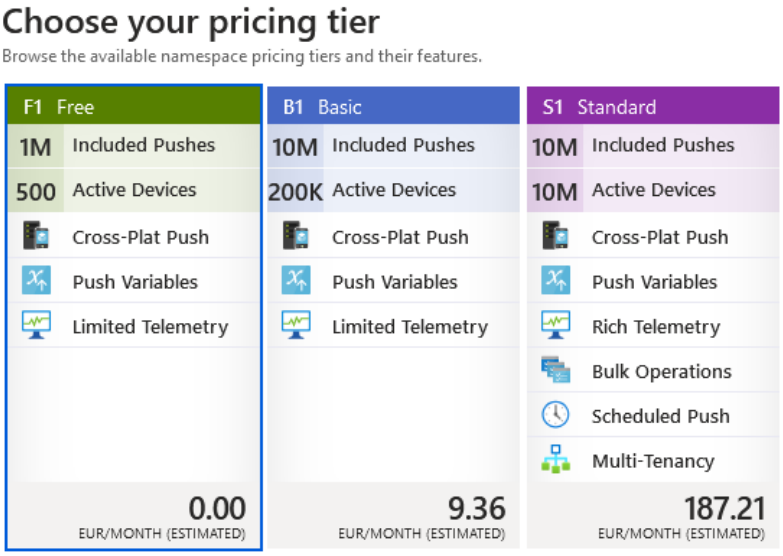

Abbildung 4.13: Tarife Azure Notification Hub. (Quelle: Microsoft 2025b)

Datenschutz (NFA 8)

Die Speicherung von Kundendaten erfolgt in der von den Nutzern ausgewählten Region. Es ist möglich, den Dienst in verschiedenen Regionen, wie beispielsweise Deutschland, zu hosten, was eine datenschutzkonforme Speicherung innerhalb der EU ermöglicht (Microsoft 2025f). Zur Sicherstellung der Sicherheit von Push-Benachrichtigungen verwendet Azure das Konzept der Shared Access Signature (SAS), das den Zugriff auf den Dienst kontrolliert (Microsoft 2023).

Portabilität (NFA 11)

Die Bereitstellung von Push-Benachrichtigungen auf mehreren Plattformen, einschließlich Android und iOS, wird von ANH unterstützt (Microsoft 2025c). Jedoch ist ANH zur Zustellung von Push-Benachrichtigungen auf die Integration verschiedener externer Dienste angewiesen. Hierzu zählt APNs für iOS-Geräte (Microsoft 2021) und FCM für Android-Geräte (Microsoft 2024b).

4.3.2 Firebase Cloud Messaging

FCM stellt einen weit verbreiteten Push-Dienst dar, der von Google betrieben wird. Er ermöglicht Entwicklern das Versenden von Push-Benachrichtigungen an eine Vielzahl von Geräten.

Verarbeitung der Daten (FA 1b)

Die Verarbeitung von Push-Benachrichtigungen wird durch FCM ermöglicht, allerdings gibt es Einschränkungen hinsichtlich der geografischen Verfügbarkeit, da der Dienst in China aufgrund der Blockierung von Google 2014 in dieser Region nicht zugänglich ist (Zheng u. a. 2020).

Integration in Anwendungen (FA 5)

Mit FCM können Nachrichten auch dann an Endgeräte zugestellt werden, wenn die zugehörige Anwendung im Hintergrund läuft oder vollständig geschlossen ist. Zudem stellt Google plattformspezifische SDKs bereit (Firebase 2025g), die eine effiziente und unkomplizierte Integration in individuelle Anwendungen ermöglichen. Darüber hinaus bietet Google eine umfassende Dokumentation an (Firebase 2024b), die die Integration und Erweiterung durch die zahlreichen bereitgestellten Funktionen der Plattform erleichtert.

Zuverlässigkeit (NFA 1)

Eine hohe Zuverlässigkeit wird durch das FCM-Backend gewährleistet, da es auf der stabilen Infrastruktur von Google basiert (Firebase 2025d). Im Falle von Problemen dauert die Behebung in der Regel zwischen einem und acht Tagen, die auf der offiziellen Firebase-Statusseite dokumentiert werden (Firebase 2025c).

Skalierbarkeit (NFA 2)

Die Plattform ist auf hohe Skalierbarkeit ausgelegt, was eine effiziente Verarbeitung einer großen Anzahl von Benachrichtigungen in kurzer Zeit ermöglicht. Die Firebase Cloud Messaging HTTP v1 API erlaubt standardmäßig 600 000 Sendeanfragen pro Minute und Projekt (Firebase 2025a). Diese Grenze wird durch einen Token-Bucket-Mechanismus gesteuert, bei dem sich das Anfragekontingent jede Minute automatisch erneuert (Firebase 2025a). Bei einer Überschreitung der Dienstkapazität kann es jedoch zu einer serverseitigen Drosselung kommen, wodurch eingehende Anfragen abgelehnt werden (Firebase 2025a). FCM ermöglicht die Ansprache von Geräten, ohne deren Akkuverbrauch signifikant zu erhöhen dank eigener in Android Geräten verbauten Technologien (Firebase 2025b). Allerdings kann es in Phasen mit erhöhtem Traffic zu temporären Verzögerungen kommen, die als Traffic-Spitzen bezeichnet werden (Firebase 2025a).

Reaktionszeiten (NFA 3)

Die Reaktionszeiten von FCM sind in der Regel sehr schnell, wobei 98 % der Nachrichten innerhalb von 500 ms zugestellt werden (Moroney 2017).

Erweiterbarkeit (NFA 4)

FCM bietet eine solide Erweiterbarkeit, da es im Google Firebase-Ökosystem integriert ist. Entwickler können die Funktionalitäten des Dienstes mit anderen Firebase-Diensten kombinieren und so erweiterte Lösungen erstellen.

Aktualität und Zukunftsfähigkeit (NFA 6)

Es liegen keine präzisen Informationen zu den Aktualisierungen von FCM vor, jedoch wird mindestens einmal jährlich ein umfangreiches Update bereitgestellt und regelmäßig kleinere Features oder Bugfixes (Firebase 2025f). Darüber hinaus ist

FCM der am weitesten verbreitete Push-Dienst. Dadurch basieren viele andere Push-Dienste auf der Infrastruktur von FCM.

Externe Dienste und Abhängigkeiten (NFA 7)

Der Dienst wird von Google bereitgestellt und ist grundsätzlich kostenlos, wobei ein Nutzungskontingent angeboten wird, das für viele Anwendungen ausreichend ist. Bei steigendem Bedarf besteht die Möglichkeit, auf den Blaze-Plan umzusteigen, der zusätzliche Funktionen sowie eine größere Flexibilität bietet (Firebase 2025e). Spezifische Lizenzinformationen für FCM sind nicht verfügbar, da der Dienst Teil des Firebase-Ökosystems von Google ist. Die Lizenzbedingungen von FCM sind daher in den allgemeinen Firebase-Nutzungsbedingungen enthalten.

Datenschutz (NFA 8)

Der Dienst erfüllt wesentliche Datenschutz- und Sicherheitsstandards (Firebase 2024a). Alle Daten werden über die Google-Server verarbeitet und gespeichert.

Portabilität (NFA 11)

Eine Vielzahl von Plattformen wird unterstützt, darunter Android, iOS sowie Webanwendungen (Firebase 2024b). Für die Nachrichtenübermittlung auf Android-Geräten mit Google Play-Diensten greift FCM auf die sogenannte Android Transport Layer (ATL) zurück (siehe Abb. A.2, S. 107). Dabei handelt es sich um eine interne Komponente der Google Play-Dienste, die speziell für die Kommunikation mit Google-Diensten wie FCM entwickelt wurde. Diese Transportschicht steht ausschließlich Google-eigenen Services zur Verfügung, sodass alternative Push-Benachrichtigungssysteme keinen Zugriff auf ATL haben (Firebase 2025b). Für Apple-Geräte erfolgt die Zustellung von Nachrichten über APNs. Im Bereich der Webanwendungen wird das standardisierte Web Push-Protokoll verwendet, welches eine breite Kompatibilität mit gängigen Webbrowsern gewährleistet.

4.3.3 OneSignal

OneSignal ist ein weit verbreiteter Push-Dienst, der Entwicklern ermöglicht, Nachrichten in kurzer Zeit an eine Vielzahl von Geräten zu senden.

Verarbeitung der Daten (FA 1b)

Die effiziente Verarbeitung von Push-Benachrichtigungen wird durch OneSignal gewährleistet, wodurch Entwicklern das Senden von Benachrichtigungen an eine Vielzahl von Geräten ermöglicht wird (OneSignal 2025c).

Integration in Anwendungen (FA 5b)

Der Versand von Benachrichtigungen ist auch dann möglich, wenn die entsprechende Anwendung im Hintergrund läuft oder bereits beendet wurde. Außerdem bietet es eine nahtlose Integration durch die Bereitstellung umfassender SDKs (OneSignal 2024c). Entwickler können die Funktionalitäten leicht in bestehende Anwendungen integrieren. Es gibt eine umfassende Dokumentation (OneSignal 2025d), wodurch Entwickler die Integration in bestehende Systeme effizient umsetzen und das Verhalten von Benachrichtigungen gezielt anpassen können.

Zuverlässigkeit (NFA 1)

Eine hohe Zuverlässigkeit wird durch OneSignal gewährleistet, was sich in den Leistungskennzahlen und den Echtzeit-Updates zur Zustellbarkeit sowie der API-Latenz widerspiegelt (OneSignal 2025f). OneSignal gibt an, über eine Million Nachrichten pro Sekunde zustellen zu können. Die Zustellrate der letzten 90 Tage liegt bei 99,98 % (Stand 11. Februar 2025, Quelle: OneSignal 2025f). Die Behebung von Fehler dauert meist nur wenige Stunden.

Skalierbarkeit (NFA 2)

OneSignal gibt an, eine erstklassige Zuverlässigkeit bei der Nachrichtenzustellung im großen Maßstab zu bieten (Adler 2022). Im kostenlosen Tarif ist der Versand auf 150 Benachrichtigungen pro Sekunde und Anwendung begrenzt, während der kostenpflichtige Plan eine Kapazität von bis zu 6 000 Nachrichten pro Sekunde und Anwendung ermöglicht (OneSignal 2025e).

Reaktionszeiten (NFA 3)

Die Reaktionszeiten von OneSignal sind im Allgemeinen sehr schnell, jedoch können Verzögerungen aufgrund von Netzwerkbedingungen oder Serverlast auftreten. Grundsätzlich können mehr als eine Million Nachrichten pro Sekunde versendet werden (Langholz 2021). Jedoch ist auch dieser Dienst von FCM und APNs abhängig.

Erweiterbarkeit (NFA 4)

Zur Erweiterbarkeit liegen keine Informationen vor.

Aktualität und Zukunftsfähigkeit (NFA 6)

Durch regelmäßige Produktaktualisierungen wird sichergestellt, dass die Plattform zukunftsfähig bleibt. Jährlich werden etwa 40 wichtige Updates durchgeführt(Adler 2022), was zeigt, dass der Dienst kontinuierlich weiterentwickelt wird. Dies stellt sicher, dass die Plattform mit den neuesten technologischen Anforderungen und Markttrends Schritt hält. OneSignal ermöglicht es mehr als einer Million Unternehmen, täglich zwölf Milliarden Nachrichten zu versenden. Etwa jede fünfte neue App nutzt das OneSignal SDK (OneSignal 2025a).

Externe Dienste und Abhängigkeiten (NFA 7)

Der Dienst wird von der Firma OneSignal, Inc. betrieben und zeichnet sich durch kontinuierliche Weiterentwicklungen aus (Adler 2022). Der Dienst stellt eine kostenlose Basisversion zur Verfügung, die grundlegende Funktionen umfasst, während erweiterte Features über kostenpflichtige Pläne ab 9 US-Dollar pro Monat zugänglich sind (Adler 2022). Bestimmte Lizenzbedingungen werden nicht festgelegt.

Datenschutz (NFA 8)

OneSignal verfolgt das Ziel, seinen Nutzern die Einhaltung der Datenschutz-Grundverordnung (DSGVO) zu ermöglichen (OneSignal 2023). Unter anderem bietet der Dienst die Möglichkeit, Nutzerdaten erst nach ausdrücklicher Zustimmung zu übermitteln. Zudem wurde eine API-Funktion zur Löschung von Nutzerdaten implementiert, ebenso wie eine vereinfachte Möglichkeit zum Export von Nutzerdaten (Deglin 2018).

Portabilität (NFA 11)

Es werden viele Plattformen unterstützt. Darunter sind auch Android, iOS und Web, wodurch eine hohe Portabilität gewährleistet wird (Adler 2022). Für Android-Geräte, die über Google Play-Dienste verfügen, erfolgt die Zustellung von Push-Benachrichtigungen über FCM (OneSignal 2024a). Dadurch besteht eine technologische Abhängigkeit von der Leistungsfähigkeit und Verfügbarkeit der Firebase-Infrastruktur. Auch Geräte des Herstellers Huawei, die keine Google Play-Dienste nutzen, werden unterstützt. Hier greift OneSignal auf den Huawei Mobile Service (HMS) und dessen Push Kit zurück (OneSignal 2024b). Für Apple-Endgeräte erfolgt die Zustellung über APNs (OneSignal 2025b), während im Webbereich das Web Push Protokoll zum Einsatz kommt (OneSignal 2024d).

4.3.4 Gegenüberstellung und Bewertung

Azure Notification Hubs, Firebase Cloud Messaging und OneSignal bieten unterschiedliche Stärken im Bereich der Push-Dienste.

Die untersuchten Dienste ermöglichen die Verarbeitung von Push-Benachrichtigungen und bieten Entwicklern eine Lösung zum Senden von Nachrichten an verschiedene Plattformen (FA 1b). Allerdings ist FCM in China nicht verfügbar, was die globale Nutzbarkeit einschränkt.

Alle drei Plattformen unterstützen die Zustellung von Benachrichtigungen selbst bei vollständig geschlossener Anwendung (FA 5a). Zudem stellen sie SDKs zur Verfügung, die die Integration in individuelle Anwendungen vereinfachen (FA 5b) und den Entwicklungsaufwand deutlich reduzieren.

In Bezug auf die Zuverlässigkeit (NFA 1) profitieren sowohl ANH als auch FCM

von der robusten Infrastruktur ihrer jeweiligen Cloud-Anbieter. ANH nutzt Verfügbarkeitszonen zur Absicherung gegen Ausfälle. Fehlerbehebungen können bei FCM einige Tage in Anspruch nehmen. OneSignal weist ebenfalls eine hohe Zuverlässigkeit auf, mit einer gemeldeten Zustellrate von 99,98 % und schnellen Reaktionszeiten bei Fehlern.

Die Skalierbarkeit (NFA 2) hinsichtlich der Nachrichtenzustellung ist bei allen drei Diensten hoch. ANH ermöglicht das Versenden von Millionen von Push-Benachrichtigungen ohne Anpassung der Architektur. FCM ist ebenfalls für eine große Nutzerzahl ausgelegt, kann aber bei Spitzenlasten Verzögerungen aufweisen. OneSignal unterstützt ebenfalls große Mengen an Nachrichten und hebt hervor, dass über eine Million Nachrichten pro Sekunde verarbeitet werden können.

Die Reaktionszeiten (NFA 3) variieren zwischen den Diensten. FCM liefert 98 % der Nachrichten innerhalb von 500 ms, während ANH aufgrund der Nutzung externer Push-Dienste keine garantierten Latenzzeiten bieten kann, aber in der Regel 99 % der Nachrichten innerhalb von einer Sekunde zustellt. OneSignal bietet ebenfalls eine schnelle Zustellung.

Hinsichtlich der Erweiterbarkeit (NFA 4) zeigen ANH und FCM klare Stärken. ANH lässt sich nahtlos in andere Azure-Dienste integrieren, während FCM eine enge Verzahnung mit dem Firebase-Ökosystem aufweist. OneSignal hingegen macht hierzu keine konkreten Angaben.

Die Aktualität und Zukunftsfähigkeit (NFA 6) wird bei ANH und FCM durch regelmäßige Updates im Rahmen ihrer Cloud-Ökosysteme sichergestellt. FCM erhält mindestens einmal jährlich ein größeres Update, während OneSignal durch häufige Produktaktualisierungen mit etwa 40 Updates pro Jahr kontinuierlich weiterentwickelt wird. Anhand der Installationszahlen der Push Notification SDKs bei Google Play und im App Store kann man erkennen, dass FCM deutlich häufiger genutzt wird als OneSignal (siehe Abb. 4.14).

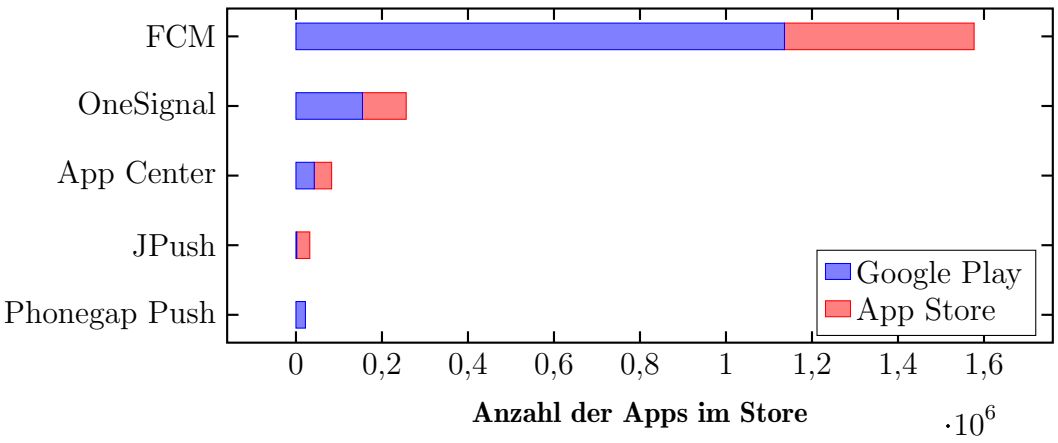

Abbildung 4.14: Am häufigsten genutzte Push-Benachrichtigungs-SDKs im Google Play Store und App Store. (Stand: 23. März 2025, Quelle: 42matters 2025a und 42matters 2025b)

Die Kostenstruktur (NFA 7a) unterscheidet sich zwischen den Diensten. ANH bietet eine kostenlose Basisversion an, die es ermöglicht, bis zu eine Million Push-Nachrichten zu versenden, sowie zusätzliche kostenpflichtige Modelle. FCM ist grundsätzlich kostenlos, mit kostenpflichtigen Optionen im nutzungsbasierten Tarif. OneSignal verfolgt ein Freemium-Modell mit einer kostenlosen Basisversion und erweiterten Funktionen ab 9 US-Dollar pro Monat. Bei FCM und OneSignal ist es möglich unbegrenzt viele Push-Benachrichtigungen kostenlos zu senden.

Beim Datenschutz (NFA 8) ermöglicht ANH eine Speicherung in verschiedenen Regionen, darunter die EU. FCM verarbeitet alle Daten über Google-Server, während OneSignal Nutzerdaten in den USA speichert und DSGVO-konforme Opt-In-Mechanismen bietet.

FCM und OneSignal bieten eine hohe Portabilität (NFA 10), da sie Push-Benachrichtigungen für Android, iOS und Web unterstützen. Während ANH keine Web Push-Benachrichtigungen ermöglicht.

Tabelle 4.3: Vergleich der Push-Dienste

Anforderung	**Azure Notification Hubs**	**Firebase Cloud Messaging**	**OneSignal**
Verarbeitung der Daten (FA 1b)	(✓) nicht in China	(✓) nicht in China	✓
Integration in Anwendungen (FA 5b)	✓	✓	✓
Zuverlässigkeit (NFA 1)	✓	✓	✓
Skalierbarkeit (NFA 2)	✓	✓	✓
Reaktionszeiten (NFA 3)	✓	✓	✓
Wartbarkeit und Erweiterbarkeit (NFA 4)	✓	✓	k.A.
Aktualität und Zukunftsfähigkeit (NFA 6)	k.A.	k.A. (weiteste Verbreitung)	ca. 40 Aktualisierungen jährlich
Kosten (NFA 7a)	1 Mio Pushes Kostenlos	Kostenlos	Kostenlos
Lizenz (NFA 7b)	k.A.	k.A.	k.A.
Datenschutz (NFA 8)	✓	✓	✓ (extra DSGVO Optionen)

Anforderung	Azure Notification Hubs	Firebase Cloud Messaging	OneSignal
Portabilität Android, iOS, Web (NFA 11)	✓, ✓, ×	✓, ✓, ✓	✓, ✓, ✓

Die Gegenüberstellung verdeutlicht, dass ANH für ein generisches Benachrichtigungssystem aufgrund potenziell hoher Kosten bei intensiver Nutzung und der fehlenden Unterstützung von Web Pushes weniger geeignet ist. Dieses Problem tritt bei FCM und OneSignal nicht auf. Allerdings ist es auch bei FCM, wie bei ANH, nicht möglich, Push-Benachrichtigungen in China zu versenden. Daher stellt OneSignal die insgesamt geeignetste Lösung dar.

4.4 Vergleich von Push-Backends

Push-Backends sind essenzielle Bestandteile moderner Softwarearchitekturen, da sie eine effiziente und zuverlässige Zustellung von Push-Benachrichtigungen ermöglichen. Sie übernehmen dabei essenzielle Aufgaben wie die Verwaltung von Nutzern und Endgeräten sowie das Speichern und Weiterleiten von Benachrichtigungen an die jeweils zuständigen Push-Dienste. In diesem Kapitel wird PushNotifier mit dem firmeninternen PushBackend der Develappers GmbH verglichen. Die Auswahl der beiden Systeme erfolgte bewusst, da derzeit nur sehr wenige eigenständige Push-Backends verfügbar sind, die für die Integration in generische Benachrichtigungssysteme infrage kommen. Viele Anbieter entscheiden sich daher dafür, eigene interne Backends zu entwickeln, um spezifische funktionale und nichtfunktionale Anforderungen gezielt erfüllen zu können.

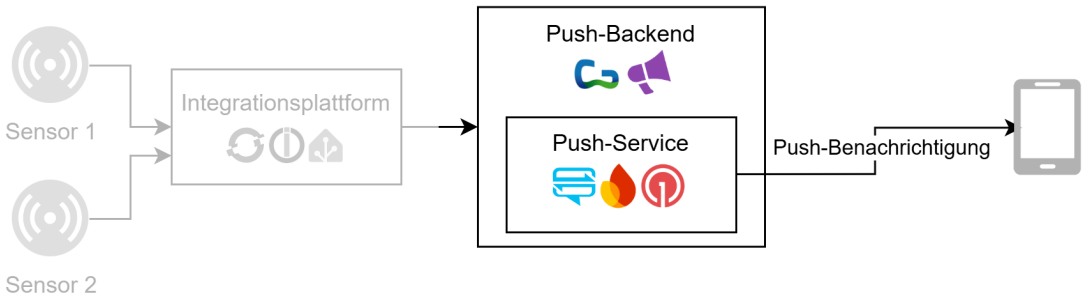

Abbildung 4.15: Push-Backends in dem generischen Benachrichtigungssystem.

Der Vergleich erfolgt anhand der aufgestellten funktionalen und nichtfunktionalen Anforderungen, um eine fundierte Bewertung hinsichtlich technischer und betrieblicher Aspekte zu ermöglichen. Jedoch werdenden einige funktionale Anforderungen nicht bewertet, da sie sich auf die Integrationsplattform oder die App beziehen (FA 1a und FA 5b). Außerdem wird die nichtfunktionale Anforderung an die Kompatibilität (NFA 10) nicht bewertet (siehe Tab A.1, S. 101). Die hier untersuchten

Backends haben bereits eine Möglichkeit implementiert, um Push-Benachrichtigungen zu versenden.

Durch diesen umfassenden Vergleich sollen die jeweiligen Stärken und Schwächen der beiden Systeme herausgearbeitet werden, um eine fundierte Entscheidungsgrundlage für deren Einsatz in dem generischen Benachrichtigungssystem zu bieten.

4.4.1 PushNotifier

PushNotifier ist ein externes Push-Backend, das es ermöglicht, kleine Nachrichten und URLs von einem PC oder über die Webseite oder die API direkt an Android- oder iOS-Geräte zu senden. Allerdings fehlen genaue Angaben oder Tests hinsichtlich der Zuverlässigkeit (NFA 1) und der Reaktionszeiten (NFA 3).

Integration in Anwendungen (FA 5b)

Die Integration in individuelle Anwendungen ist möglich (PushNotifier 2023b). Dadurch können Benachrichtigungen per Push versendet werden (FA 1b). Jedoch ist das Empfangen der Nachrichten in einer eigenen Anwendung nicht möglich. Die Anwendung ist mandantenfähig, wobei die Nutzerverwaltung (FA 3) ausschließlich über die Website (PushNotifier 2025f) erfolgt. Auch die Verwaltung von Geräten muss über die Website erfolgen (PushNotifier 2025e), da die API nur eine GET-Methode zur Anzeige der Geräte bereitstellt, jedoch keine Möglichkeit bietet, neue Geräte hinzuzufügen oder bestehende zu bearbeiten (PushNotifier 2023a). Außerdem scheint es keine Funktion zum Abruf des Benachrichtigungsverlaufes (FA 2) zu geben.

Skalierbarkeit (NFA 2)

Es gibt keine konkreten Angaben zur Skalierbarkeit bei einer großen Anzahl von Nutzern und Benachrichtigungen. Zwar können unbegrenzt viele Geräte hinzugefügt und Nachrichten versendet werden, jedoch kann ein hohes Nachrichtenaufkommen innerhalb eines Accounts zu Verzögerungen oder zum Ausbleiben von Benachrichtigungen führen (PushNotifier 2025c).

Wartbarkeit und Erweiterbarkeit (NFA 4)

Da der Quellcode von PushNotifier nicht öffentlich zugänglich ist, besteht keine Möglichkeit, Erweiterungen am Dienst vorzunehmen oder diesen direkt zu warten. Dies schränkt die Flexibilität ein, da Anpassungen oder Änderungen an der Infrastruktur nur durch den Anbieter selbst durchgeführt werden können.

Ressourcennutzung (NFA 5)

Da PushNotifier vom Anbieter selbst bereitgestellt wird, sind keine eigenen Ressourcen für die Bereitstellung und den Betrieb des Dienstes erforderlich. Die gesamte

Infrastruktur und Wartung werden zentral vom Anbieter übernommen.

Aktualität und Zukunftsfähigkeit (NFA 6)

Es liegen weder Informationen zu regelmäßigen Updates noch Hinweise auf aktive Diskussionsforen oder Community-Plattformen zu PushNotifier vor. Lediglich ein einzelner Beitrag in einem allgemeinen Forum stellt das Produkt vor. Dies deutet darauf hin, dass PushNotifier derzeit nur von einer begrenzten Anzahl an Nutzern verwendet wird, was Fragen zur langfristigen Zukunftssicherheit aufwirft.

Externe Dienste (NFA 7)

PushNotifier ist in seiner Funktionsweise auf externe Dienste angewiesen, insbesondere auf FCM für Android-Geräte und APNs für iOS-Geräte (PushNotifier 2025b). Jedoch ist die Nutzung des Dienstes komplett kostenlos (NFA 7a), jedoch wird keine Angabe zur Lizenz (NFA 7b) gemacht.

Datenschutz (NFA 8)

Es wird lediglich beschrieben, welche Daten gespeichert und verarbeitet werden (PushNotifier 2025b), aber nicht, dass dies gemäß bestimmten Sicherheitsstandards geschieht.

Benutzerfreundlichkeit (NFA 9)

Die Benutzeroberfläche ist ansprechend gestaltet und intuitiv verständlich. Sie ermöglicht eine unkomplizierte Anmeldung, das Versenden von Benachrichtigungen sowie die Verwaltung von Geräten (siehe Abb. 4.16 und Abb. 4.17).

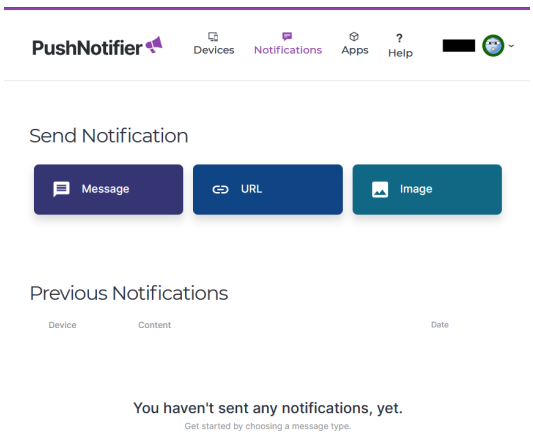

Abbildung 4.16: Senden einer Benachrichtigung mittels PushNotifier. Quelle: (PushNotifier 2025g)

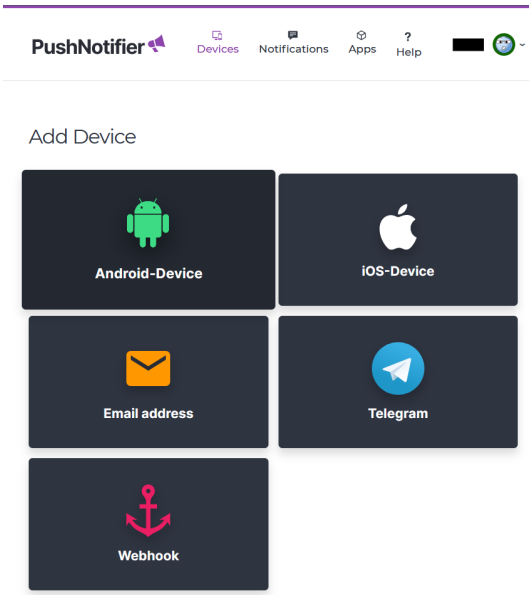

Abbildung 4.17: Hinzufügen eines Gerätes mittels PushNotifier. (Quelle: PushNotifier 2025a)

Portabilität (NFA 11)

PushNotifier unterstützt die Zustellung von Push-Benachrichtigungen für iOS- und Android-Geräte, jedoch keine Web-Push-Benachrichtigungen (PushNotifier 2025d).

4.4.2 PushBackend

Das PushBackend ist eine App, die Push-Kampagnen verwaltet und automatisierte Push-Nachrichten an mobile Geräte sendet.

Integration in Anwendungen (FA 5b)

Das PushBackend verfügt über eine API, die eine umfassende Verwaltung ermöglicht. Über diese Schnittstelle können nicht nur Push-Nachrichten versendet (FA 1b), sondern auch der Benachrichtigungsverlauf abgerufen (FA 2) sowie Nutzer (FA 3) und Geräte (FA 4) verwaltet werden. Es steht zwar kein SDK zur Verfügung, das den Empfang von Nachrichten in Individualanwendungen direkt ermöglicht, jedoch kann die FCM-Konfiguration problemlos an das Backend übermittelt werden, wodurch der Empfang von Nachrichten in der App realisiert wird. Dadurch bietet das System eine flexible und zentrale Steuerung aller relevanten Funktionen.

Zuverlässigkeit (NFA 1)

Die Zuverlässigkeit des PushBackends hängt maßgeblich von den verwendeten externen Diensten wie Azure und FCM ab. Allerdings stellt das System sicher, dass keine Nachrichten verloren gehen, da alle versendeten Benachrichtigungen gespeichert werden.

Skalierbarkeit (NFA 2)

Die Skalierbarkeit hinsichtlich der Nachrichtenanzahl und der Nutzeranzahl des PushBackends wird maßgeblich durch die Abhängigkeit von FCM sowie die gewählte Azure-Pricing-Tier bestimmt. Da Azure je nach Tarif unterschiedliche Rechenkapazitäten bereitstellt, kann die Leistungsfähigkeit des Systems variieren. Eine höhere Skalierbarkeit erfordert entsprechend eine Anpassung des gebuchten Ressourcenpakets.

Reaktionszeiten (NFA 3)

Die Reaktionszeiten des PushBackends werden ebenfalls durch die Leistung von Azure und die Integration mit FCM beeinflusst. Da die Rechenkapazität je nach gewähltem Azure-Tarif variiert, kann dies die Geschwindigkeit der Nachrichtenverarbeitung und Nachrichtenzustellung beeinflussen.

Wartbarkeit und Erweiterbarkeit (NFA 4)

Das PushBackend zeichnet sich durch eine hohe Wartbarkeit und Erweiterbarkeit aus, da es auf ASP .NET basiert. Diese Architektur ermöglicht eine modulare Entwicklung, erleichtert die Integration neuer Funktionen und unterstützt eine langfristige Wartung durch standardisierte Frameworks und Werkzeuge.

Ressourcennutzung (NFA 5)

Durch die optimale Anpassung an die zugrunde liegende Infrastruktur arbeitet das PushBackend ressourceneffizient. Mit steigender Auslastung des Systems nimmt auch der Ressourcenverbrauch entsprechend zu.

Aktualität und Zukunftsfähigkeit (NFA 6)

Da es sich um eine firmeninterne Lösung handelt, kann die Aktualität und Zukunftsfähigkeit flexibel an die unternehmensspezifischen Anforderungen angepasst werden. Weiterentwicklungen und Updates liegen vollständig in der eigenen Hand, wodurch eine langfristige Wartung und Optimierung sichergestellt werden kann.

Externe Dienste (NFA 7)

Die Kosten (NFA 7a) des PushBackends entstehen durch das Hosting auf Azure, wobei insbesondere der App Service Plan und die Datenbank monatliche Ausgaben verursachen. Diese Kosten variieren je nach gewähltem Pricing-Tier, da unterschiedliche Leistungsstufen unterschiedliche Preise für Rechenleistung und Speicherkapazität mit sich bringen. Die Kosten für den App Service Plan belaufen sich auf null bis zu 185 Euro im Monat (Microsoft 2025a). Zusätzlich fallen monatliche Kosten für die benötigte MS SQL-Datenbank an, die je nach Leistungsstufe zwischen 4 und 420 Euro betragen können. Die Gesamtkosten des PushBackends hängen somit von den spezifischen Anforderungen an Rechenleistung, Speicherplatz und Skalierbarkeit ab (Microsoft 2025g). Da es sich um eine firmeninterne Lösung handelt, unterliegt das PushBackend keiner spezifischen Lizenz (NFA 7b). Die Firma besitzt die vollständigen Rechte an der Software und kann sie beliebig nutzen, anpassen und weiterentwickeln, ohne Einschränkungen durch externe Lizenzbestimmungen.

Datenschutz (NFA 8)

Personenbezogene Daten werden vom PushBackend nicht gespeichert, wodurch datenschutzrechtliche Anforderungen unterstützt werden. Allerdings können im Payload der versendeten Nachrichten theoretisch beliebige Inhalte hinterlegt werden, einschließlich sensibler Informationen. Da die Anwendung auf Azure bereitgestellt wird, werden alle relevanten Sicherheitsstandards von Microsoft eingehalten, um die Vertraulichkeit und Integrität der verarbeiteten Daten zu gewährleisten.

Benutzerfreundlichkeit (NFA 9)

Eine dedizierte Benutzeroberfläche für individuelle Nutzer wird vom PushBackend nicht bereitgestellt. Stattdessen erfolgt die Registrierung der Geräte automatisch über die App. Benachrichtigungen werden über HTTPS von einer Integrationsplattform an das Backend übermittelt.

Portabilität (NFA 11)

Plattformübergreifender Einsatz ermöglicht dem PushBackend den Versand von Push-Benachrichtigungen an Android-, iOS- und Web-Anwendungen. Dadurch ist eine flexible Nutzung in verschiedenen Systemumgebungen möglich.

4.4.3 Gegenüberstellung und Bewertung

PushNotifier und das firmeninterne PushBackend weisen unterschiedliche Stärken im Bereich der Backend-Lösungen für Push-Benachrichtigungen auf.

Während PushNotifier die Integration in Anwendungen (FA 5b) und den Versand von Push-Benachrichtigungen (FA 1b) ermöglicht, erfolgt die Nutzer- (FA 3) und Geräteverwaltung (FA 4) ausschließlich über die Website. Das PushBackend bietet eine umfassende API. Diese ermöglicht nicht nur den Versand von Benachrichtigungen, sondern auch deren Empfang und den Abruf des Benachrichtigungsverlaufes (FA 2). Zusätzlich erlaubt die API die direkte Verwaltung von Nutzern und Geräten. Ein SDK zum Einbinden der Benachrichtigungen gibt es in beiden Anwendungen nicht, jedoch kann bei dem PushBackend die FCM-Konfiguration übermittelt werden, um den Empfang in der App zu realisieren.

PushNotifier macht keine Angaben zur Zuverlässigkeit des Dienstes (NFA 1). Bei dem PushBackend hingegen hängt sie von externen Diensten wie Azure und Firebase ab.

Hinsichtlich der Skalierbarkeit (NFA 2) erlaubt PushNotifier unbegrenzt viele Geräte und Nachrichten, was jedoch bei hohem Nachrichtenaufkommen zu Verzögerungen oder Ausfällen führen kann. Eine Angabe zu der maximalen Anzahl an Nutzern gibt es nicht. Im Gegensatz dazu ist die Skalierbarkeit des PushBackends durch Azure und FCM steuerbar, wodurch eine gezielte Leistungssteigerung möglich ist. Dies macht das PushBackend verlässlicher, da Nachrichten nicht aufgrund von Überlastung abgelehnt werden.

PushNotifier gibt keine spezifischen Angaben zu den Reaktionszeiten (NFA 3). Jedoch sind beide Dienste von FCM und APNs abhängig, was zu unvorhersehbaren Schwankungen in der Geschwindigkeit führen kann. Das PushBackend ist zudem noch von Azure abhängig, wobei die Rechenkapazität je nach Azure-Tarif variiert. Diese Flexibilität bietet Anpassungsmöglichkeiten, kann aber auch zu Verzögerungen oder Ausfällen führen, wenn nicht ausreichend Ressourcen gebucht sind.

Die Wartbarkeit und Erweiterbarkeit (NFA 4) sind bei PushNotifier eingeschränkt, da der Quellcode nicht öffentlich zugänglich ist und Anpassungen nur durch den Anbieter vorgenommen werden können. Im Gegensatz dazu bietet das PushBackend dank seiner ASP .NET-Architektur eine hohe Flexibilität, ermöglicht eine modulare Entwicklung und unterstützt eine langfristige Wartung durch standardisierte Frameworks und Werkzeuge.

PushNotifier benötigt keine eigenen Ressourcen (NFA 5), da der Dienst vom Anbieter bereitgestellt und gewartet wird. Das PushBackend hingegen nutzt die zugrunde liegende Infrastruktur so effizient wie möglich, um eine ressourcenschonende Arbeitsweise zu gewährleisten.

Die langfristige Zukunftssicherheit (NFA 6) von PushNotifier bleibt fraglich, da es keine Anzeichen für regelmäßige Updates oder eine aktive Community gibt. Das PushBackend hingegen bietet durch seine firmeninterne Lösung eine flexible Anpassung an unternehmensspezifische Anforderungen, was langfristige Wartung und kontinuierliche Optimierung gewährleistet.

Die Nutzung externer Dienste (NFA 7) wie FCM und APNs für iOS ist für PushNotifier erforderlich. Das PushBackend setzt ebenfalls auf diese Dienste und verursacht noch zusätzliche Hosting-Kosten auf Azure, die je nach gewähltem Tarif variieren. Als firmeninterne Lösung bietet es den Vorteil, dass keine Lizenzgebühren anfallen.

Im Hinblick auf den Datenschutz (NFA 8) beschreibt PushNotifier die gespeicherten Daten, ohne jedoch auf die Einhaltung von Sicherheitsstandards einzugehen. Das PushBackend hingegen garantiert durch die Einhaltung der Sicherheitsstandards von Microsoft Azure einen besseren Schutz der Daten.

Um die Benutzerfreundlichkeit zu gewährleisten (NFA 9), bietet PushNotifier eine ansprechend gestaltete und intuitive Benutzeroberfläche, die eine einfache Anmeldung und Verwaltung ermöglicht. Im Gegensatz dazu stellt das PushBackend keine dedizierte Benutzeroberfläche für Nutzer bereit und ermöglicht die automatische Registrierung der Geräte über die Individualanwendung.

PushNotifier unterstützt die Zustellung von Push-Benachrichtigungen für iOS- und Android-Geräte, jedoch keine Web-Push-Benachrichtigungen (NFA 11). Das PushBackend hingegen ist plattformübergreifend einsetzbar und ermöglicht das Versenden von Push-Benachrichtigungen an Android-, iOS- und Web-Anwendungen, was eine flexible Nutzung in verschiedenen Systemumgebungen erlaubt.

Tabelle 4.4: Vergleich der Push-Backends

Anforderung	**PushNotifier**	**Develappers PushBackend**
API/ SDK (FA 5b)	✓/ ×	✓/ ×
Verarbeitung der Daten (FA 1b)	✓	✓
Benachrichtigungsverlauf (FA 2)	×	✓
Nutzerverwaltung/ Mandantenfähigkeit (FA 3)	Registrierung nur über Login auf der Seite möglich	✓
Geräteverwaltung (FA 4)	nur GET; SET über Website	✓
Zuverlässigkeit (NFA 1)	k.A.	Abhängig von Azure und FCM
Skalierbarkeit (NFA 2)	✓	FCM Limit, Azure pricing tier
Reaktionszeiten (NFA 3)	k.A.	Abhängig von Azure und FCM
Wartbarkeit und Erweiterbarkeit (NFA 4)	×	✓
Ressourcennutzung (NFA 5)	Keine	effizient
Aktualität und Zukunftsfähigkeit (NFA 6)	×	firmenintern
Externe Dienste (NFA 7)	Fcm, Apple Push	Fcm, Azure
Kosten (NFA 7a)	Kostenlos	Selbsthosting
Lizenz (NFA 7b)	k.A.	✓
Datenschutz (NFA 8)	k.A.	✓ Azure
Benutzerfreundlichkeit (NFA 9)	✓	✓
Portabilität Android, iOS, Web (NFA 11)	✓, ✓, ×	✓, ✓, ✓

Zusammenfassend lässt sich feststellen, dass das PushBackend der Firma Develappers GmbH aufgrund seiner großen Flexibilität und der Möglichkeit zur langfristigen Wartung besonders gut geeignet ist, die Anforderungen an ein generisches

Benachrichtigungssystem zu erfüllen. Im Vergleich dazu zeigt PushNotifier in mehreren Bereichen Schwächen, insbesondere bei den funktionalen Anforderungen, der Zuverlässigkeit, der Integration externer Dienste sowie der Zukunftsfähigkeit. Diese Einschränkungen machen PushNotifier weniger geeignet für größere oder langfristige Projekte.

5 Konzeption und prototypische Umsetzung

Die Entwicklung eines zuverlässigen und skalierbaren Benachrichtigungssystems erfordert eine fundierte konzeptionelle Planung sowie eine erste prototypische Umsetzung. In diesem Kapitel wird die Systemarchitektur detailliert beschrieben, um eine robuste Grundlage für die Integration von Sensordaten in ein generisches Benachrichtigungssystem zu schaffen.

Aufbauend auf der Analyse geeigneter Technologien erfolgt die Auswahl der Systemkomponenten, darunter die Integrationsplattform, das Push-Backend und der Push-Dienst. Anschließend wird die prototypische Umsetzung beschrieben, die als Grundlage für die spätere vollständige Implementierung dient. Ziel ist es, ein flexibles System zu entwickeln, das Sensordaten zuverlässig verarbeitet und zeitnah an die Nutzer übermittelt.

5.1 Systemarchitektur

Im vorliegenden Kapitel wird die Systemarchitektur des generischen Benachrichtigungssystems für Individualanwendungen detailliert vorgestellt. Diese Architektur bildet die Grundlage für die Integration von Sensorereignissen in eine Push-Benachrichtigungsarchitektur, in der Daten aus unterschiedlichen Sensorquellen verarbeitet und unmittelbar an die Nutzer übermittelt werden. Ziel ist es, ein zuverlässiges, skalierbares und benutzerfreundliches System zu entwickeln, das die Interaktion zwischen Anwendern und Anwendungen durch zeitnahe Ereignisbenachrichtigungen optimiert. Bei der Konzeption der Systemarchitektur wurden sowohl die funktionalen als auch die nichtfunktionalen Anforderungen berücksichtigt.

Um eine Sensoränderung als Push-Benachrichtigung in einer Anwendung zu empfangen, ist ein mehrstufiger Aufbau empfehlenswert, da er eine hohe Kompatibilität und Flexibilität bei der Integration unterschiedlicher Systeme ermöglicht. Ein möglicher erster Schritt in einem solchen Aufbau ist der Einsatz einer Integrationsplattform, die verschiedene Sensoren in einem gemeinsamen System zusammenführt (siehe Abb. 5.1, S. 54). Beispiele hierfür sind openHAB, ioBroker oder Home Assistant. Diese Plattformen unterstützen je nach System eine Vielzahl an Kommunikationsprotokollen und ermöglichen die Anbindung an etablierte Standards wie MQTT, Zigbee, Z-Wave und LoRaWAN. Dadurch lässt sich eine flexible Integration unterschiedlichster Sensoren realisieren und eine zentrale Grundlage für die Erfassung und Verarbeitung relevanter Sensordaten schaffen.

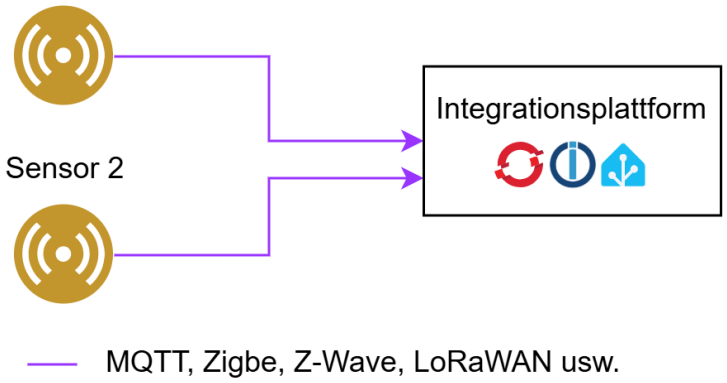

Abbildung 5.1: Übermittlung der Sensordaten zur Integrationsplattform.

Anschließend müssen die von den Sensoren erfassten Daten an ein Benachrichtigungssystem weitergeleitet werden. Hierzu existieren grundsätzlich zwei Ansätze (siehe Abb. 5.2). Einerseits kann eine individuelle Erweiterung für die Integrationsplattform entwickelt werden, die die Sensordaten an ein dediziertes Backend wie das Develappers PushBackend oder PushNotifier übermittelt. Andererseits kann auf bereits etablierte Systeme wie Pushsafer, Pushover, Pushbullet oder Gotify zurückgegriffen werden, die häufig über fertige Adapter zur Integration in Integrationsplattformen verfügen. In diesem Fall wird das vorhandene Backend des jeweiligen Systems genutzt. Beiden Ansätzen ist gemein, dass die jeweilige Erweiterung die Informationen per HTTPS überträgt und eine Regel benötigt, die festlegt, unter welchen Bedingungen sie ausgelöst wird.

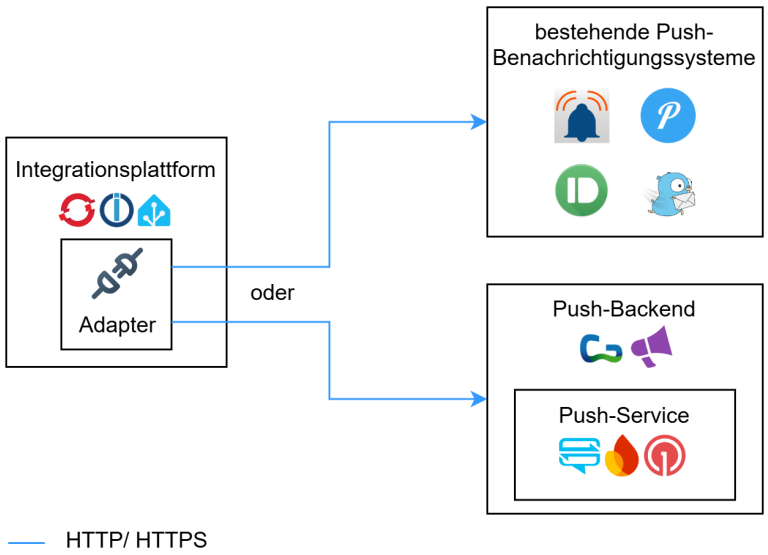

Abbildung 5.2: Weiterleitung der Daten an ein Benachrichtigungssystem oder ein Push-Backend.

Unabhängig von der gewählten Herangehensweise übernimmt ein integrierter Push-Dienst, beispielsweise ANH, FCM oder OneSignal, die Aufgabe, die Nachricht an die Zielgeräte zu senden. Auf der Empfängerseite muss eine individuell entwickelte App installiert sein, die in der Lage ist, die eintreffenden Push-Benachrichtigungen zu verarbeiten und dem Nutzer in geeigneter Weise darzustellen (siehe Abb. 5.3).

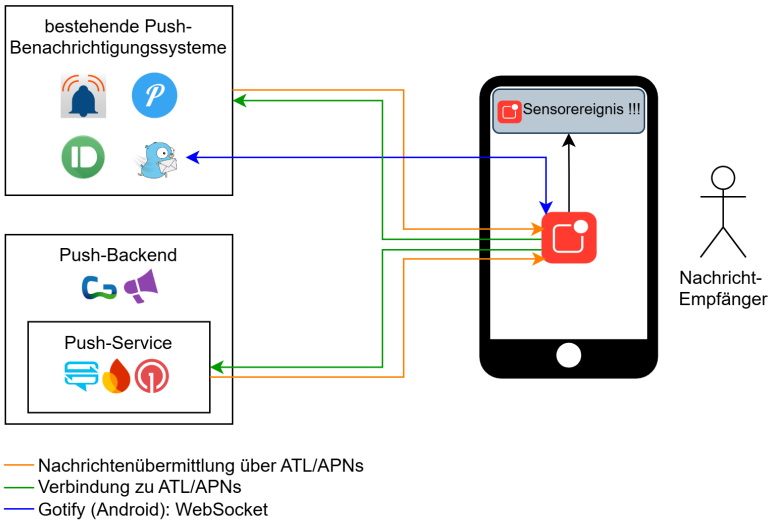

Abbildung 5.3: Empfangen der Benachrichtigung in der App und Darstellung als Push-Benachrichtigung.

Durch diesen modularen Aufbau wird ein flexibles und skalierbares Benachrichtigungssystem realisiert (siehe Abb. 5.4, S. 56), das sich an unterschiedliche Einsatzszenarien anpassen lässt. Jede Komponente, von der Integrationsplattform über das Backend bis hin zum Push-Dienst, kann isoliert optimiert oder bei Bedarf ausgetauscht werden, sodass das Gesamtsystem den spezifischen Anforderungen gerecht wird.

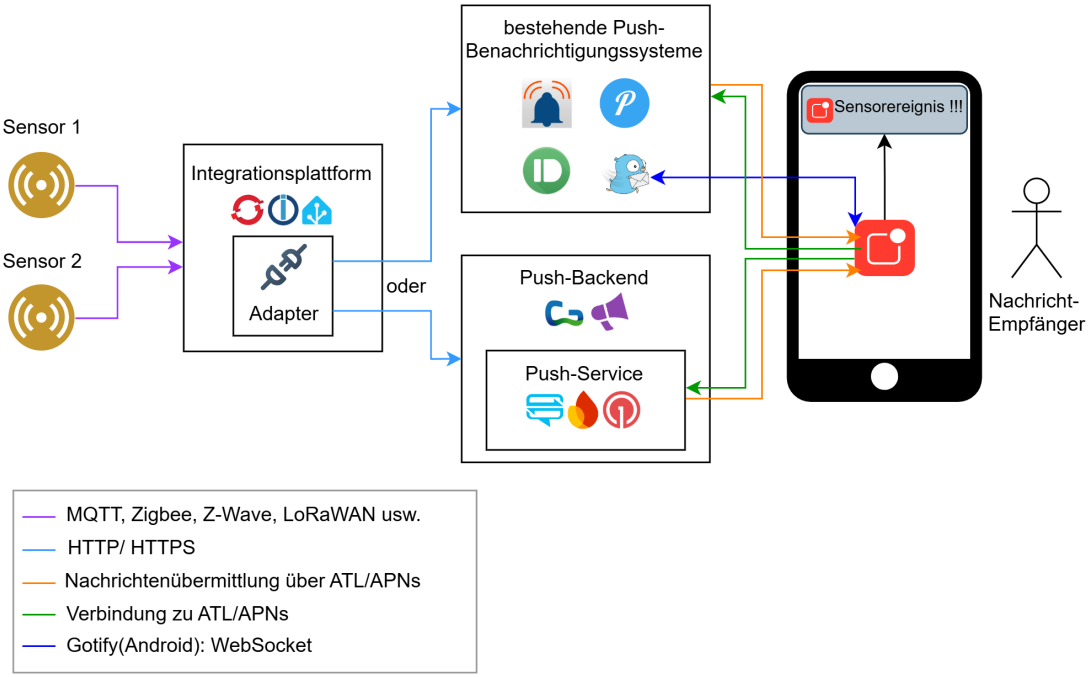

Abbildung 5.4: Systemarchitektur für ein generisches Benachrichtigungssystem.

5.2 Auswahl der Technologien basierend auf der Analyse

Die Auswahl geeigneter Technologien ist ein entscheidender Schritt für die erfolgreiche Umsetzung eines generischen Benachrichtigungssystems. Dabei müssen die zuvor aufgestellten Anforderungskriterien (siehe Kapitel 3.3, S. 6) und Vergleiche berücksichtigt werden. In diesem Abschnitt werden die getroffenen technologischen Entscheidungen begründet.

Da bereits im vorherigen Kapitel die Systemarchitektur detailliert beschrieben wurde, folgt hier eine gezielte Auswahl der Technologien für die jeweiligen Komponenten.

Die Integrationsplattform dient als Schnittstelle zur Integration und Verwaltung von Sensoren. Zur Auswahl standen ioBroker, openHAB und Home Assistant. Wie auch in der Tabelle 4.1 Seite 20 dargestellt eigenen sich alle der drei Plattformen für diesen Anwendungsfall und erfüllen alle Anforderungen. Während Home Assistant eine hohe Verbreitung aufweist und besonders für Einsteiger attraktiv ist, fiel die Entscheidung aufgrund vorhandener Erfahrung in der Programmiersprache JavaScript auf ioBroker. Die Implementierung soll jedoch langfristig auch für openHAB und Home Assistant erfolgen, um eine breitere Nutzerbasis zu erreichen.

Für das Push-Backend wurden das interne PushBackend der Develappers GmbH und PushNotifier gegenübergestellt (siehe Tab 4.4, S. 51). Aufgrund der hohen Anpassungs- und Erweiterbarkeit fiel die Entscheidung auf das Develappers-Backend, da es sowohl alle funktionalen als auch nichtfunktionalen Anforderungen erfüllt und

flexibel an spezifische Bedürfnisse angepasst werden kann. Der einzige Nachteil dieser Lösung besteht in der Notwendigkeit eines eigenen Hostings, wodurch zusätzliche Kosten entstehen. Die Nutzung eines bestehenden Systems hätte den Vorteil gehabt, dass für alle drei Integrationsplattformen bereits Erweiterungen existieren. Allerdings konnte keines der untersuchten Systeme sämtliche Anforderungen vollständig erfüllen, und die Anpassung von Gotify hätte einen hohen Aufwand erfordert.

Bei dem Vergleich der Push-Dienste wurden ANH, FCM und OneSignal analysiert (siehe Tab 4.3, S. 43). Während ANH potenziell hohe Kosten verursacht und OneSignal zusätzliche Integrationsarbeit erfordert, wurde FCM als bevorzugte Lösung gewählt, da es bereits im PushBackend integriert ist und eine zuverlässige sowie kosteneffiziente Zustellung von Benachrichtigungen ermöglicht.

Die gewählte Kombination aus ioBroker, dem Develappers PushBackend und FCM (siehe Abb. 5.5) bildet eine stabile und effiziente Grundlage für das Benachrichtigungssystem. Durch die geplante Erweiterung auf weitere Integrationsplattformen kann eine hohe Flexibilität und breite Nutzeransprache sichergestellt werden.

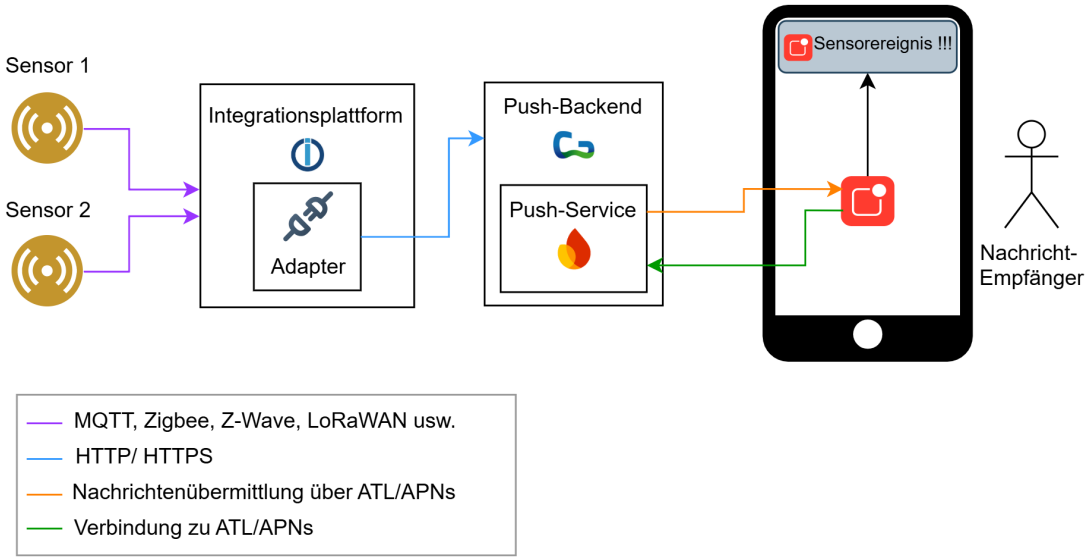

Abbildung 5.5: Technologieauswahl für ein generisches Benachrichtigungssystem.

5.3 Implementierung des Systems

Im folgenden Abschnitt wird ein prototypisches Benachrichtigungssystem implementiert, das als exemplarischer Nachweis für die technische Realisierbarkeit des Konzepts dient.

5.3.1 Aufbau des Prototyps in der Testumgebung

Für die Realisierung des Systems wird ioBroker benötigt. Diese Integrationsplattform kann sowohl lokal als auch in der Cloud betrieben werden, wobei die von ioBroker zur Verfügung gestellte Cloud-Variante kostenpflichtig ist. Alternativ lässt sich ioBroker auf einer Vielzahl von Plattformen und Betriebssystemen installieren, darunter Windows, Linux, OSX, Raspberry Pi, ARM und PC (ioBroker 2024a). Während lokale Installationen den Vorteil bieten, unabhängig von einer Internetverbindung zu funktionieren und hinsichtlich des Datenschutzes sicherer zu sein, ermöglichen Cloud-Lösungen einen ortsunabhängigen Zugriff, sind jedoch mit fortlaufenden Kosten verbunden. Für die Entwicklung des Prototyps wurde die Installation von ioBroker in einem Docker-Container gewählt. Diese Methode ermöglicht eine effiziente Bereitstellung, bietet eine hohe Flexibilität und ist besonders für Testzwecke geeignet.

Zur Simulation von Ereignissen innerhalb von ioBroker wird ein Sensor benötigt, der Zustandsänderungen an das System überträgt. Um die Funktionsweise des Prototyps zu überprüfen, wurde in ioBroker ein neuer State definiert, über den die Funktionalität von Sensoren wie Lichtschaltern oder Feuermeldern simuliert werden kann.

Für die Verarbeitung und Zustellung von Benachrichtigungen wird das PushBackend der Develappers GmbH eingesetzt. Die Bereitstellung dieses Dienstes setzt eine MS SQL-Datenbank sowie einen HTTPS-Webserver voraus. Das Hosting erfolgt in Microsoft Azure, da hier der Standort Deutschland gewählt werden kann und Azure zusätzlich erweiterte Sicherheitsmechanismen bereitstellt.

Die Push-Benachrichtigungen werden über das bereits in das Backend integrierte FCM verarbeitet, sodass keine weiteren Konfigurationsschritte erforderlich sind. Zur Überprüfung der Zustellung dient eine speziell entwickelte Test-App für Android-Geräte.

5.3.2 Entwicklung und Integration einer Erweiterung in die Integrationsplattform

Um Sensordaten effizient zu erfassen, zu verarbeiten und an das Develappers Push-Backend zu übermitteln, wird eine maßgeschneiderte Erweiterung für die Plattform ioBroker entwickelt. Dafür wird ein spezieller Adapter implementiert, der die nahtlose Integration ermöglicht und auf die spezifischen Anforderungen der Systemarchitektur abgestimmt ist. Der folgende Abschnitt beschreibt die technische Umsetzung dieser Lösung im Detail.

Anforderungen an den Adapter

1. Weiterleitung von Sensorbenachrichtigungen an das PushBackend
   Der Adapter sollte dazu befähigt sein, eintreffende Sensorereignisse zuverlässig zu erfassen und mit minimaler Latenz über das HTTP- oder HTTPS-Protokoll an einen definierten Zielendpunkt zu übertragen. Dieser Endpunkt, `SendToUser` des Develappers PushBackends, dient der Weiterleitung der Benachrichtigungen an die Nutzer.

2. Nutzeridentifikation über API Key
   Jede Kommunikation mit dem PushBackend erfordert eine eindeutige Identifikation des Nutzers. Dies geschieht über einen nutzerspezifischen API Key, der in den Adaptereinstellungen konfigurierbar sein muss. Durch diese Maßnahme wird sichergestellt, dass nur autorisierte Benutzer Benachrichtigungen empfangen und Sicherheitsrisiken minimiert werden.

3. Anpassungsmöglichkeiten für Titel und Nachricht
   Ein wesentliches Merkmal des Adapters ist die Möglichkeit zur individuellen Anpassung von Benachrichtigungstiteln und der Nachricht (siehe FA 1c). Nutzer sollen in der Lage sein, den Inhalt der Meldungen entsprechend ihren Bedürfnissen zu modifizieren, um eine bessere Kontextualisierung der Sensorwerte zu erreichen.

4. Definierbare Regeln für Benachrichtigungen
   Nutzer benötigen eine flexible Möglichkeit, Regeln für Benachrichtigungen zu definieren, um festzulegen, unter welchen Bedingungen eine Meldung generiert wird. Dazu sind verschiedene Konfigurationsoptionen erforderlich, darunter die Auswahl spezifischer Sensoren oder Sensorgruppen, die Festlegung von Schwellenwerten oder Ereignisauslösern sowie die individuelle Gestaltung von Titeln und Nachrichten je nach Ereignis.

5. Ständige Verfügbarkeit des Adapters
   Um eine sofortige Reaktion auf Sensoränderungen zu gewährleisten, muss der Adapter im `always on`-Modus betrieben werden. Dies bedeutet, dass er permanent im Hintergrund läuft und eingehende Sensorwerte kontinuierlich überprüft. Eine verzögerte Reaktion könnte dazu führen, dass kritische Sensorinformationen zu spät an den Nutzer weitergeleitet werden.

Umsetzung des Adapters

Der Adapter wurde mithilfe des Assistenten zur Erstellung neuer ioBroker-Adapter generiert, wodurch eine standardisierte Projektstruktur sowie grundlegende Konfigurationsdateien automatisch bereitgestellt wurden. Beim ersten testweisen Hochladen des Adapters in den ioBroker Docker Container wurde deutlich, dass eine Anpassung notwendig war, da die zugrunde liegende Vorlage auf eine veraltete Version der Admin-Oberfläche verwies (siehe Listing A.2 Z. 33, S. 104). Nach der Aktualisierung des `admin`-Eintrags konnte die Entwicklung ohne weitere Einschränkungen fortgesetzt werden.

Im nächsten Schritt wurde der Konstruktor für den Adapter erstellt, wobei Eventhandler für die Zustände `ready`, `unload` und `message` definiert wurden (siehe Listing A.1 Z. 11 bis 13, S. 102).

Der `ready`-Handler wird automatisch aufgerufen, sobald eine Instanz des Adapters in ioBroker gestartet wird. In dieser Methode erfolgt eine Überprüfung, ob ein API-Token hinterlegt wurde. Falls kein Token vorhanden ist, wird eine entsprechende Warnung im ioBroker-Protokoll ausgegeben.

Abbildung 5.6: Warnung, dass der Adapter noch nicht konfiguriert wurde.

Der `unload`-Handler dient dazu, beim Stoppen des Adapters oder einer Instanz relevante Informationen oder Fehler auszugeben.

Die wichtigste Methode ist der `message`-Handler, der auf eingehende Nachrichten reagiert. Er übernimmt die Verarbeitung von Anfragen und sendet einen POST-Request an das PushBackend, um eine neue Push-Nachricht zu generieren. Das Backend übernimmt anschließend die Zustellung der Nachricht an den Nutzer. Bei erfolgreicher Übermittlung gibt der API-Aufruf einen HTTP-Response-Code 200 zurück (siehe Abb. 5.7). Im Fehlerfall wird je nach Ursache ein entsprechender Fehlercode zurückgesendet, um eine gezielte Fehlerbehandlung zu ermöglichen.

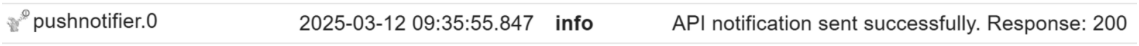

Abbildung 5.7: Erfolgreiches Senden der Benachrichtigung an das Backend.

Anpassung an die Anforderungen und Überprüfung

Im folgenden Abschnitt werden die zuvor formulierten Anforderungen an den Adapter überprüft. Es wird erläutert, inwieweit diese Anforderungen erfüllt wurden und wie die jeweilige Umsetzung erfolgte.

Die Weiterleitung von Sensorbenachrichtigungen an das PushBackend (Anforderung 1), wurde umgesetzt, indem der Adapter einen Eventhandler (`message`-Handler) für die Nachricht definiert hat. Sobald dieser Handler aktiviert wird, werden die Daten, wie oben beschrieben, an das PushBackend weitergeleitet. Diese Anforderung wurde somit vollständig erfüllt.

Die Anforderungen 2, Nutzeridentifikation über API-Token, und 3, Anpassungsmöglichkeiten für Titel und Nachricht, wurden ebenfalls erfolgreich umgesetzt. Hierfür wurde ein zusätzlicher Tab für den Adapter erstellt. Ein Adapter wird dann als Instanz hinzugefügt. In dieser Instanz können verschiedene Konfigurationen vorgenommen werden, wie etwa das Setzen des API-Tokens, des Titels und des Textes. Zur Umsetzung dieser Anpassungsmöglichkeiten wurde eine HTML-Datei erstellt,

die das Layout dieses Tabs definiert und die notwendigen Eingabefelder bereitstellt (siehe Abb. 5.8).

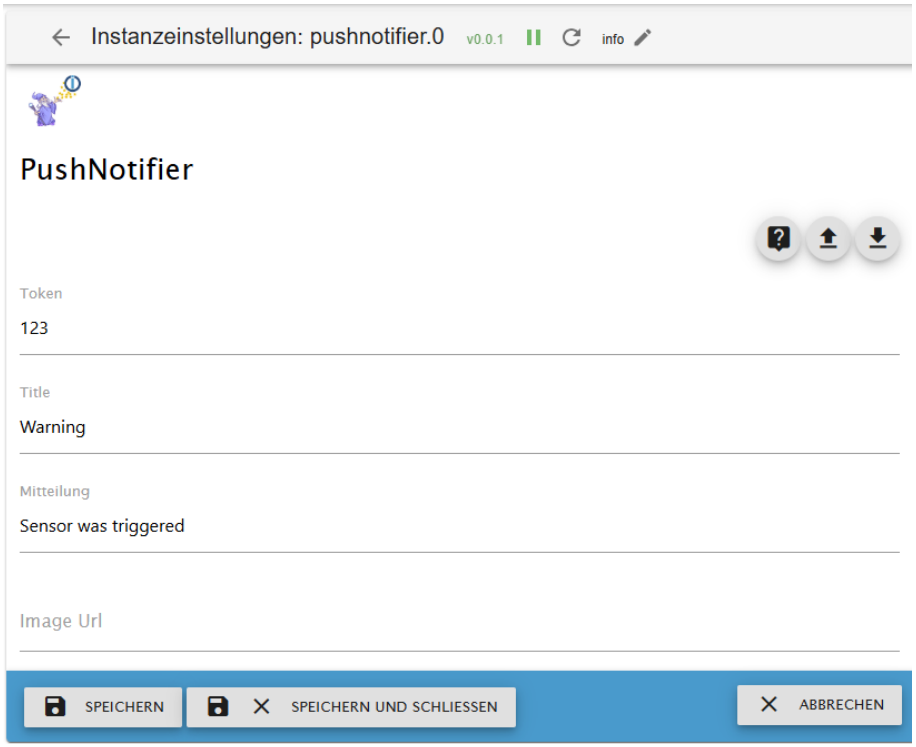

Abbildung 5.8: Instanzeinstellungen des entwickelten Adapters in ioBroker.

Um die Werte, die in der Konfiguration der Instanz festgelegt werden, in der `onMessage`-Methode verwenden zu können, wurden die entsprechenden Properties der Eingabefelder in der Konfigurationsdatei definiert (siehe Listing A.2 Z. 41 bis 45, S. 104). Diese Werte können dann direkt ausgelesen und in der `onMessage`-Methode verwendet werden (siehe Listing A.1 Z. 38, S. 102).

Die definierbaren Regeln für Benachrichtigungen (Anforderung 4) sind mithilfe des Adapters Skriptausführung möglich. Hierbei kann der Nutzer ein neues Skript hinzufügen und aus den vier verfügbaren Sprachen wählen: Rules, Blockly, JavaScript und TypeScript.

Am Beispiel von JavaScript wird ein Event Listener registriert, der auf Änderungen eines bestimmten Datenpunkts reagiert. In folgendem Fall überwacht das Skript einen Feuermelder und prüft, ob dessen Status auf wahr wechselt (siehe Listing 5.1, S. 62). Falls dies eintritt, wird über die Funktion `sendTo` eine Nachricht an die PushNotifier-Instanz des Adapters gesendet. Diese enthält sowohl einen Titel als auch eine Nachricht und wird anschließend an das Backend weitergeleitet, um die Benachrichtigung an die vorgesehenen Empfänger zu übermitteln (siehe Abb. A.3, S. 108). Die Implementierung in TypeScript folgt dem gleichen Prinzip (siehe Listing A.3, S. 106).

Listing 5.1: Beispiel eines Ausführungsscriptes in JavaScript

```
1 on({id: '0_userdata.0.Feuermelder', change: 'ne'}, function (
    obj) {
2     if (obj.state.val === true) {
3         sendTo("pushnotifier.1", "send", {
4             title: "Feuermeldung!",
5             message: "Der Feuermelder wurde ausgeloest."
6         });
7     }
8 });
```

Um den Einstieg für weniger technikaffine Nutzer zu erleichtern, wurde ein benutzerdefinierter Blockly-Block für diesen Adapter entwickelt (siehe Abb. 5.9). Der entwickelte Block ist in der Abbildung als schwarzes Element hervorgehoben. Blockly ist ein visueller Programmiereditor von Google, der auf einer Drag-and-Drop-Oberfläche basiert (Google 2025). Mit diesem speziellen Block können Nutzer direkt eine ihrer Instanzen aus einer Dropdown-Liste auswählen und optional den Titel sowie den Text individuell anpassen. Werden diese Felder nicht ausgefüllt, übernimmt der Adapter automatisch die entsprechenden Werte aus der Konfiguration der jeweiligen Instanz.

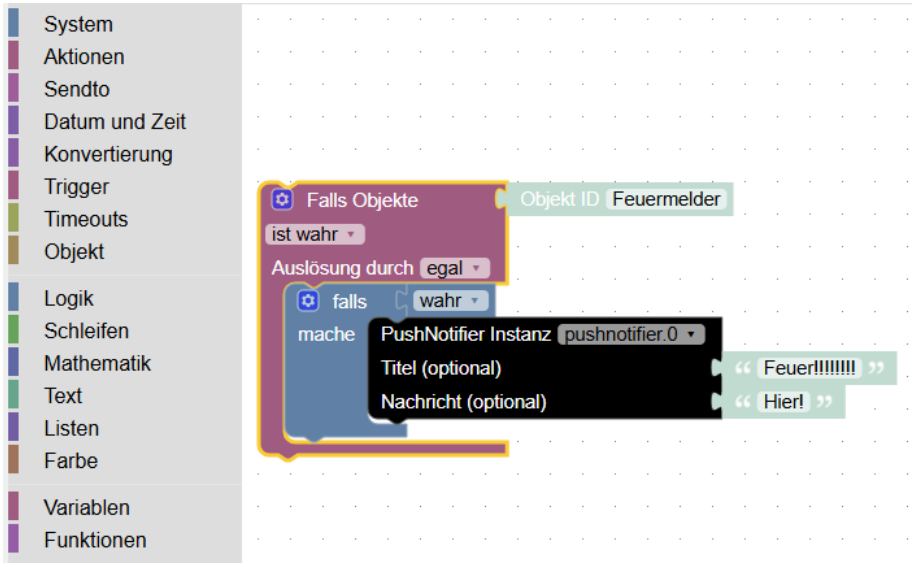

Abbildung 5.9: Blockly Block für den Adapter.

Die Sprache Rules ist grundsätzlich die einfachste Option. Allerdings bietet sie keine native SendTo-Funktion, sodass dieser Teil als separater Code eingefügt werden muss (siehe Abb. A.1, S. 106). Dadurch ist für diesen Anwendungsfall Blockly für die meisten Benutzer vermutlich die benutzerfreundlichste Lösung.

Um die ständige Verfügbarkeit des Adapters sicherzustellen (Anforderung 5), muss dieser dauerhaft im Betrieb bleiben. Dies erfordert den Einsatz des sogenannten Daemon-Modus (siehe Listing A.2 Z. 16, S. 104), der eine ununterbrochene Ausführung des Adapters gewährleistet.

6 Bewertung des implementierten Systems

Die Bewertung des implementierten Systems dient dazu, die Reaktionsgeschwindigkeit, Zuverlässigkeit und Praxistauglichkeit der entwickelten Lösung kritisch zu analysieren. Dabei wird untersucht, inwieweit die definierten Anforderungen erfüllt wurden, welche Leistungsmerkmale das System aufweist und wie es sich im Vergleich zu bestehenden Lösungen verhält. Abschließend werden mögliche Optimierungspotenziale aufgezeigt, um zukünftige Weiterentwicklungen gezielt zu ermöglichen.

6.1 Erfüllung der Anforderungen

Die Überprüfung der Erfüllung der Anforderungen stellt sicher, dass das entwickelte System den in der Anforderungsanalyse definierten funktionalen und nichtfunktionalen Kriterien entspricht. Dazu wird analysiert, inwieweit die gesetzten Ziele erreicht wurden und ob das Benachrichtigungssystem flexibel genug ist, um in verschiedenen Individualanwendungen eingesetzt zu werden.

6.1.1 Funktionale Anforderungen

In diesem Abschnitt wird bewertet, inwieweit das entwickelte Benachrichtigungssystem die funktionalen Anforderungen erfüllt.

Die Verarbeitung der Sensordaten (FA 1) wird durch das System unterstützt, indem eine flexible Architektur bereitgestellt wird, die eine effektive Regelung und Automatisierung von Prozessen ermöglicht. In ioBroker können definierte Ereignisse erkannt, Regeln erstellt und durch den implementierten Adapter eine Benachrichtigung an das Backend ausgelöst werden (FA 1a). Diese Benachrichtigung wird anschließend über Develappers PushBackend und FCM an den Nutzer übermittelt (FA 1b). Die Benachrichtigungen enthalten dabei den Titel, den Text und den Zeitstempel des Ereignisses (FA 1c).

Die Speicherung der Benachrichtigungen (FA 2) erfolgt durch das Develappers PushBackend, welches alle gesendeten Benachrichtigungen speichert und zur späteren Abrufbarkeit bereitstellt. Dadurch wird eine vollständige Historie der Benachrichtigungen erstellt, die dem Nutzer jederzeit zur Verfügung steht.

Die Mandantenfähigkeit (FA 3) wird sowohl durch ioBroker als auch durch das

Develappers PushBackend unterstützt. Beide Systeme ermöglichen die Verwaltung mehrerer unabhängiger Mandanten. Dabei bleiben deren Daten und Konfigurationen strikt voneinander getrennt, obwohl dieselbe Infrastruktur genutzt wird. Diese Eigenschaft ist im Gesamtsystem vollständig umgesetzt. Mehrere Nutzer können es somit parallel und unabhängig voneinander verwenden, ohne dass es zu Konflikten zwischen ihren Daten oder Einstellungen kommt.

Das Develappers PushBackend übernimmt die Geräteverwaltung (FA 4), indem es die Registrierung, Verwaltung und Löschung der Endgeräte des Nutzers ermöglicht, die Push-Benachrichtigungen empfangen. Jedes Endgerät wird eindeutig identifiziert, was eine gezielte Zustellung von Benachrichtigungen ermöglicht.

Die Integration in Anwendungen (FA 5) wird durch das Develappers PushBackend und FCM ermöglicht. Über die API des Backends können Funktionen wie das Abrufen der letzten Benachrichtigungen und des API-Tokens genutzt werden. Darüber hinaus können sich Nutzer einloggen und registrieren. Die Integration von FCM ist ebenfalls möglich, wobei aktuell kein SDK für eine einfache Einbindung in andere Anwendungen existiert (FA 5b). Die Test-App ist jedoch in der Lage, Push-Benachrichtigungen zuverlässig zu empfangen und anzuzeigen, unabhängig davon, ob die Anwendung aktiv oder im Hintergrund läuft (FA 5a).

Zusammenfassend lässt sich festhalten, dass das Gesamtsystem die funktionalen Anforderungen in den Bereichen Sensordatenübermittlung, Benachrichtigungserstellung, Speicherung, Mandantenfähigkeit, Geräteverwaltung und Integration in Anwendungen umfassend erfüllt. Die einzelnen Systemkomponenten tragen jeweils zur vollständigen Umsetzung der definierten Anforderungen bei und gewährleisten so die Entwicklung eines funktional robusten Systems.

6.1.2 Nichtfunktionale Anforderungen

In diesem Abschnitt wird das entwickelte Benachrichtigungssystem im Hinblick auf die nichtfunktionalen Anforderungen untersucht. Dabei wird analysiert, inwieweit die einzelnen Systemkomponenten diese Anforderungen erfüllen, um darauf aufbauend bewerten zu können, ob das Gesamtsystem den definierten Anforderungen insgesamt entspricht.

Das System gewährleistet eine zuverlässige Zustellung (NFA 1). Während für die Stabilität von ioBroker selbst keine offiziellen Angaben vorliegen, bildet das FCM-Backend auf den Google-Servern eine verlässliche Grundlage. Sollte der FCM-Dienst vorübergehend nicht erreichbar sein, übernimmt das PushBackend eine erneute Zustellung der Benachrichtigung. Dadurch wird sichergestellt, dass Nutzer unabhängig von temporären Ausfällen jederzeit über relevante Ereignisse informiert bleiben.

Das Benachrichtigungssystem ist hinsichtlich der Skalierbarkeit (NFA 2) so ausgelegt, dass es eine wachsende Anzahl gleichzeitiger Nutzer und Sensoren verarbeiten

kann, ohne die Performance signifikant zu beeinträchtigen. Für ioBroker selbst liegen hierzu zwar keine offiziellen Angaben vor, jedoch erlaubt das FCM-Backend bis zu 600 000 Sendeanfragen pro Minute und ist somit ein limitierender Faktor. Darüber hinaus kann das PushBackend flexibel an unterschiedliche Lastanforderungen angepasst werden, da es auf Azure-Diensten basiert und durch die Auswahl geeigneter Pricing-Tiers skalierbar bleibt.

Die Anforderung an niedrige Reaktionszeiten (NFA 3) wird durch das verzögerungsarme Auslösen von Benachrichtigungen erfüllt. Das PushBackend ist abhängig vom FCM-Dienst, der etwa 98 % der Nachrichten innerhalb von 0,5 s zustellt. Dadurch wird sichergestellt, dass Benachrichtigungen spätestens zehn Sekunden nach Erfassung eines Sensorereignisses ausgelöst werden (NFA 3b).

Das System ist so konzipiert, dass Updates und Erweiterungen (NFA 4) ohne signifikante Unterbrechungen des Betriebs durchgeführt werden können. Die modulare Architektur ermöglicht eine effiziente Wartung und eine problemlose Integration zusätzlicher Funktionen. Durch die Open-Source-Basis von ioBroker steht eine breite Auswahl an Erweiterungen in Form von Adaptern zur Verfügung, während das PushBackend ebenfalls eine gezielte Wartung und Anpassung ermöglicht. Auch FCM bietet Erweiterungsmöglichkeiten.

Das Benachrichtigungssystem nutzt Ressourcen effizient (NFA 5), da das PushBackend ressourcenschonend arbeitet und FCM als Cloud-Dienst keine zusätzlichen Anforderungen an das lokale System stellt. Für ioBroker liegen keine offiziellen Angaben vor.

Das System basiert auf modernen, weit verbreiteten Technologien, die kontinuierlich weiterentwickelt werden. Dies gewährleistet eine langfristige Anpassungsfähigkeit an neue technologische Standards (NFA 6). Trotz eines Rückgangs in der Nutzung von ioBroker bleibt die grundlegende Systemarchitektur zukunftssicher.

Das Benachrichtigungssystem setzt ausschließlich auf kostenlose externe Dienste (NFA 7), da ioBroker selber gehostet wird. Allerdings kann das Develappers PushBackend bei hoher Last zusätzliche Kosten verursachen, da es von Hosting-Gebühren abhängig ist. Es wurden ausschließlich Dienste ausgewählt, die eine kommerzielle Nutzung ermöglichen.

Die Einhaltung aktueller Sicherheitsstandards aller Komponenten (NFA 8) stellt den Schutz personenbezogener Daten sicher.

Das Benachrichtigungssystem bietet durch ioBroker eine intuitive Benutzeroberfläche (NFA 9), die sowohl für technisch versierte als auch für weniger erfahrene Nutzer einfach zu bedienen ist. Die Test-App zeigt den API-Token an, den Nutzer lediglich in ioBroker eintragen müssen, wodurch der Einrichtungsprozess erheblich vereinfacht wird.

Da ioBroker mit einer Vielzahl von Sensoren und Kommunikationsprotokollen kompatibel ist (NFA 10), erstreckt sich diese Kompatibilität auch auf das gesamte Benachrichtigungssystem. Dies ermöglicht eine flexible Integration unterschiedlichster Hardware-Komponenten.

Durch die Nutzung von FCM kann das System Push-Benachrichtigungen auf verschiedenen Plattformen wie iOS, Android und Web senden (NFA 11). Dies stellt eine hohe Portabilität und Geräteunabhängigkeit sicher.

Zusammenfassend erfüllt das Benachrichtigungssystem alle nichtfunktionalen Anforderungen. Die modulare und flexible Architektur in Kombination mit modernen Technologien gewährleistet ein leistungsstarkes und zukunftssicheres System. Schwachstellen liegen in den nicht vermeidbaren Kosten durch externe Dienste sowie im fehlenden SDK, das eine einfache Integration in weitere Anwendungen erschweren kann.

6.1.3 Flexibilität und Anwendbarkeit

Ein entscheidender Faktor bei der Bewertung des entwickelten Benachrichtigungssystems ist seine Fähigkeit, die definierten Anwendungsfälle und Zielgruppen effektiv zu unterstützen.

Ein wesentlicher Vorteil des Systems liegt in der hohen Kompatibilität mit einer Vielzahl von Sensoren und Ereignisquellen. Durch die Nutzung von ioBroker als Integrationsplattform können unterschiedlichste Sensoren integriert und das System flexibel an spezifische Anforderungen angepasst werden. Dies ermöglicht den Einsatz in verschiedensten Bereichen, darunter Smart-Home-Anwendungen, industrielle Automatisierung, das Gesundheitswesen und die Umweltüberwachung. Insbesondere Unternehmen und Hersteller von IoT-Geräten profitieren von dieser Offenheit, da sie eigene Sensoren nahtlos einbinden können.

Allerdings ist das System auf FCM angewiesen, um Benachrichtigungen an die Nutzer zu übermitteln. Da FCM als externer Dienst agiert, kann es in Zeiten hoher Auslastung zu Verzögerungen in der Zustellung kommen. Diese Verzögerungen liegen außerhalb der direkten Kontrolle des Systems und sind durch die Infrastruktur von FCM bedingt, die möglicherweise nicht alle Nachrichten sofort verarbeiten kann. Insbesondere für zeitkritische Anwendungen kann dies eine Herausforderung darstellen.

Hinsichtlich der Benutzerfreundlichkeit bietet das System eine hohe Flexibilität, da es sich in unterschiedliche Anwendungen mit verschiedenen Benutzeroberflächen integrieren lässt. Dadurch kann die Darstellung und Bedienung an die spezifischen Bedürfnisse der jeweiligen Zielgruppe angepasst werden.

Ein weiterer Vorteil liegt in der bewussten Kostenminimierung, die das System so-

wohl für kleinere als auch für größere Projekte attraktiv macht. Allerdings besteht für App-Entwickler der Nachteil, dass Firebase manuell konfiguriert werden muss, da kein vorgefertigtes SDK für das Benachrichtigungssystem existiert. Dennoch bietet die API des PushBackends eine umfassende Verwaltung von Nutzern, Geräten und Benachrichtigungen, wodurch langfristig eine effiziente Nutzung sichergestellt wird.

Zusammenfassend erfüllt das System die gestellten Anforderungen und erweist sich als vielseitige Lösung für die definierten Anwendungsfälle und Zielgruppen.

6.2 Leistungsanalyse

Die Leistungsfähigkeit des Systems wird maßgeblich durch Latenzzeiten und Skalierbarkeit bestimmt. Eine geringe Latenz ist entscheidend für eine schnelle Informationsverarbeitung, während eine gute Skalierbarkeit sicherstellt, dass das System auch bei hoher Last stabil bleibt. Dieses Kapitel untersucht, wie schnell das System auf Anfragen reagiert und inwieweit es sich an steigende Anforderungen anpassen kann. Der Akkuverbrauch konnte aufgrund fehlerhafter Anzeigen der Testgeräte nicht zuverlässig erfasst werden.

6.2.1 Latenzzeiten

Latenz bezeichnet die Zeitverzögerung zwischen einer Aktion und der darauf folgenden Reaktion eines Systems. Besonders in zeitkritischen Anwendungen kann selbst eine geringe Verzögerung die Funktionalität erheblich beeinflussen. Im vorliegenden Benachrichtigungssystem ist es entscheidend, die Zeitspanne zwischen einem auslösenden Sensorereignis und der Zustellung der Push-Benachrichtigung so gering wie möglich zu halten, wie in Anforderung NFA 3 definiert. Dieses Kapitel widmet sich der systematischen Erfassung und Analyse der Latenzzeiten, um mögliche Optimierungspotenziale und Störungsfaktoren aufzuzeigen.

Zur Messung der Latenz wurde das Test-Setup gemäß der Beschreibung in Kapitel 5.3 Implementierung des Systems Seite 57 aufgebaut. Die Messungen erfolgten unter unterschiedlichen Netzwerkbedingungen und zu verschiedenen Tageszeiten. Ziel war es, die Zeitspanne zwischen dem Auslösen des Sensors, der Übermittlung der Nachricht durch ioBroker an das Backend sowie der anschließenden Weiterleitung an FCM zu erfassen. Zusätzlich wurde erfasst, wann die Benachrichtigung schließlich in der App eintraf.

Messmethodik und Einflussfaktoren

Zur Nachvollziehbarkeit wurden pro Messung drei Zeitstempel erfasst: der Zeitpunkt der Sensorauslösung, die Weiterleitung der Nachricht an FCM durch das Backend und die Ankunft in der App (siehe Abb. 6.1, S. 68).

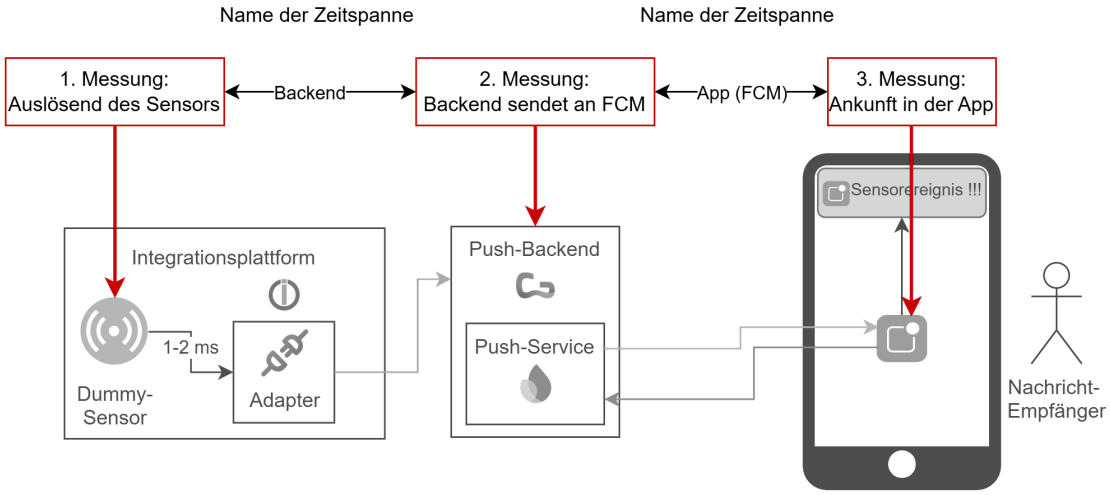

Abbildung 6.1: Messpunkte zur Bestimmung der Latenzzeiten des Prototyps.

Da die ermittelten Werte im Millisekunden- bis Sekundenbereich liegen, ist eine präzise Zeitsynchronisation der beteiligten Systeme essenziell. Zur Gewährleistung einer synchronen Zeitbasis zwischen dem ioBroker-System und dem Testgerät wurde die Übereinstimmung der Uhrzeiten mithilfe der Website time.is überprüft. Etwaige Differenzen wurden in die Berechnungen einbezogen, wobei dennoch eine potenzielle Abweichung von bis zu 0,035 s verbleiben kann. Die Zeitsynchronisation der Azure-Server, auf denen das Backend läuft, konnte hingegen nicht direkt überprüft werden. Es wird jedoch davon ausgegangen, dass die Server eine präzise Zeitquelle verwenden und keine signifikanten Abweichungen aufweisen.

Da es sich bei dem Sensor um einen Dummy-Sensor in ioBroker handelte, war die Zeitspanne zwischen der Sensorauslösung und dem Versand der Anfrage an das Backend vernachlässigbar gering. Die Verzögerung lag maximal bei zwei Millisekunden, weshalb sie in der weiteren Analyse nicht berücksichtigt wurde.

Ein weiterer Einflussfaktor auf die Zustellzeit ist das Empfangsgerät, auf dem sich die App befindet. In den durchgeführten Tests kam ein Google Pixel 7a mit Android 15 zum Einsatz, dessen Hard- und Softwarearchitektur die Latenz zusätzlich beeinflussen kann. Unterschiedliche Geräte und Betriebssystemversionen könnten zu abweichenden Ergebnissen führen.

Ein relevanter Faktor ist der Zustand der Anwendung, ob sie geöffnet, minimiert oder geschlossen ist. Zudem ist es relevant, ob das Empfangsgerät den Bildschirm eingeschaltet hatte oder ein Wake-up-Ereignis erforderlich war. Da Messungen im Zustand einer geschlossenen App nicht durchführbar waren, wurden alle Tests unter der Voraussetzung durchgeführt, dass die App sich im Vordergrund befand.

Des Weiteren ist die effektive Datenübertragungsrate dynamisch und unterliegt temporären Schwankungen, primär bedingt durch die aktuelle Netzwerkauslastung.

Messungen

Es wurden insgesamt 40 Messungen durchgeführt (siehe Tab A.2, S. 110 und Tab A.3, S. 111). Das System, auf dem ioBroker in einem Docker-Container betrieben wurde, war während aller Tests mit einem WLAN verbunden, das eine Download-Geschwindigkeit von 129 Mbit/s und eine Upload-Geschwindigkeit von 44 Mbit/s aufwies, also einer relativ hohen Datenübertragungsrate. Die gemessenen Latenzzeiten für die Kommunikation zwischen dem Sensor und dem Backend bis zur Weiterleitung an FCM lagen zwischen 0,233 s und 0,515 s. Das arithmetische Mittel dieser Stichprobe betrug 0,348 s bei einer Standardabweichung von 0,051 s.

Zu Beginn erfolgten die Messungen unter optimalen Netzwerkbedingungen (siehe Abb. 6.2). In den ersten zehn Testläufen war das Empfangsgerät mit demselben WLAN verbunden wie das System, das ioBroker im Docker-Container ausführt. Für die restlichen zehn Messungen wurde das Gerät in ein 5G-Mobilfunknetz gewechselt, das eine Download-Geschwindigkeit von 60 Mbit/s und eine Upload-Geschwindigkeit von sechs Mbit/s hatte. Dieser Wechsel des Netzwerks hätte potenziell Einfluss auf die Verzögerung zwischen FCM und dem Eingang der Nachricht in der App haben können. In den Messungen ließ sich jedoch kein signifikanter Unterschied feststellen. Gemessen wurde die Zeitspanne zwischen dem Auslösen des Sensors und dem Eintreffen der Benachrichtigung in der App. Innerhalb der erhobenen Stichprobe variierten die beobachteten Latenzzeiten zwischen 0,659 s und 1,549 s. Das arithmetische Mittel der Zustellzeiten lag bei 0,956 s, bei einer Standardabweichung von 0,251 s (siehe Tab A.4, S. 112).

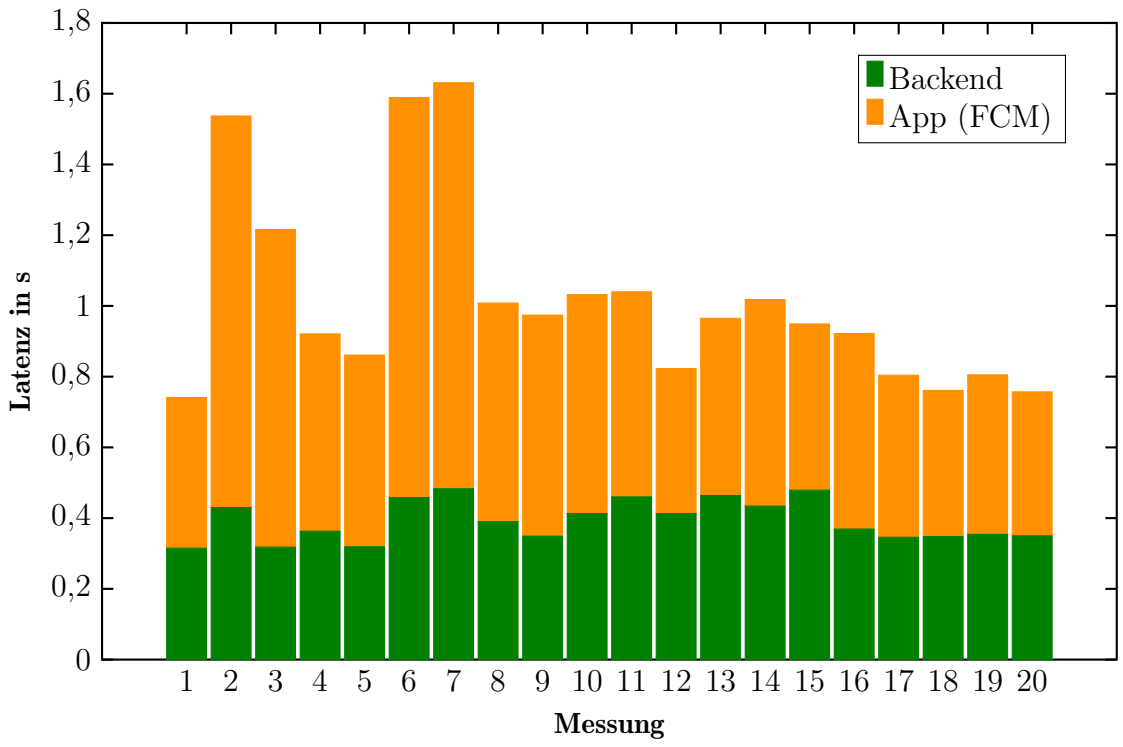

Abbildung 6.2: Latenzzeiten bei guter Netzwerkverbindung.

Hierbei lag die Latenz zwischen der Übermittlung der Nachricht vom Backend an FCM und dem Eintreffen auf dem Mobilgerät im Bereich von 0,407 s bis 1,148 s (siehe orangene Punkte in Abb. 6.3, S. 70). Einige der Tests wurden gezielt zu Stoßzeiten durchgeführt, in denen ein erhöhtes Nachrichtenaufkommen bei FCM erwartet wurde. Dennoch konnte auch hier kein klarer Zusammenhang zwischen der Tageszeit und der Latenz festgestellt werden. Das in Abbildung 6.3 Seite 70 rot gekennzeichnete arithmetische Mittel der Latenz lag bei 0,625 s, wobei die Standardabweichung der Stichprobe 0,244 s betrug.

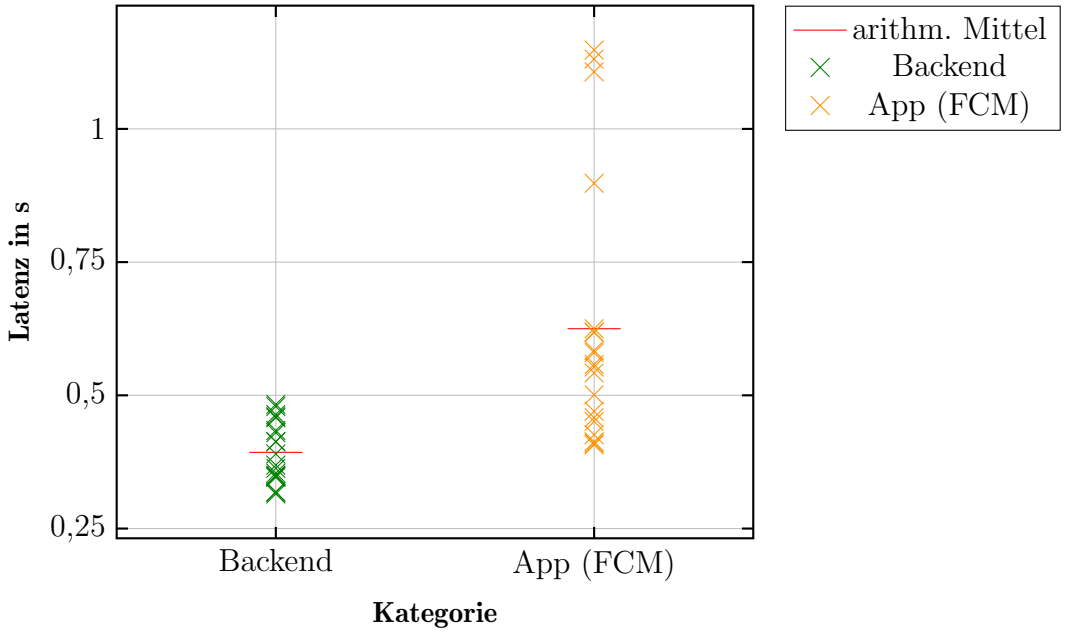

Abbildung 6.3: Verteilung der Latenzzeiten bei guter Netzwerkverbindung.

Anschließend wurden 20 weitere Messungen durchgeführt, bei denen sich das Empfangsgerät in einem 2G-Mobilfunknetz mit einer Download-Geschwindigkeit von 0,09 Mbit/s und einer Upload-Geschwindigkeit von 0,04 Mbit/s befand. Infolge dessen verlängerte sich die Gesamtzeit zwischen der Auslösung des Sensors und dem Empfang der Benachrichtigung auf bis zu 7,214 s (Abb. 6.4, S. 71), was einer Erhöhung des Maximalwertes um etwa 366 % entspricht.

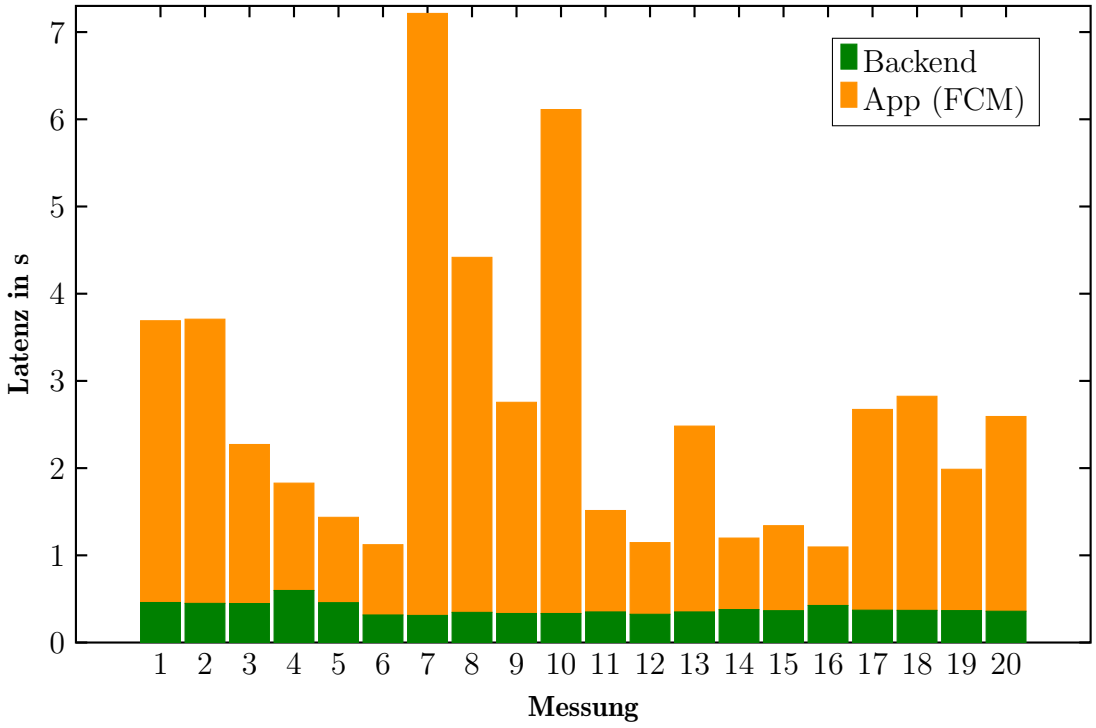

Abbildung 6.4: Erfasste Latenzzeiten des Systems unter Bedingungen eingeschränkter Netzwerkverbindung auf Empfängerseite.

Hier zeigte sich ein deutlicher Unterschied in der Zustellzeit der Push-Benachrichtigungen in der App. Bei dieser langsamen Netzwerkverbindung variierten die Latenzzeiten zwischen Backend und der App erheblich, lagen jedoch durchweg auf einem deutlich höheren Niveau zwischen 0,671 s und 6,903 s (siehe Abb. 6.5). Das arithmetische Mittel betrug 2,651 s, bei einer Standardabweichung von 1,675 s.

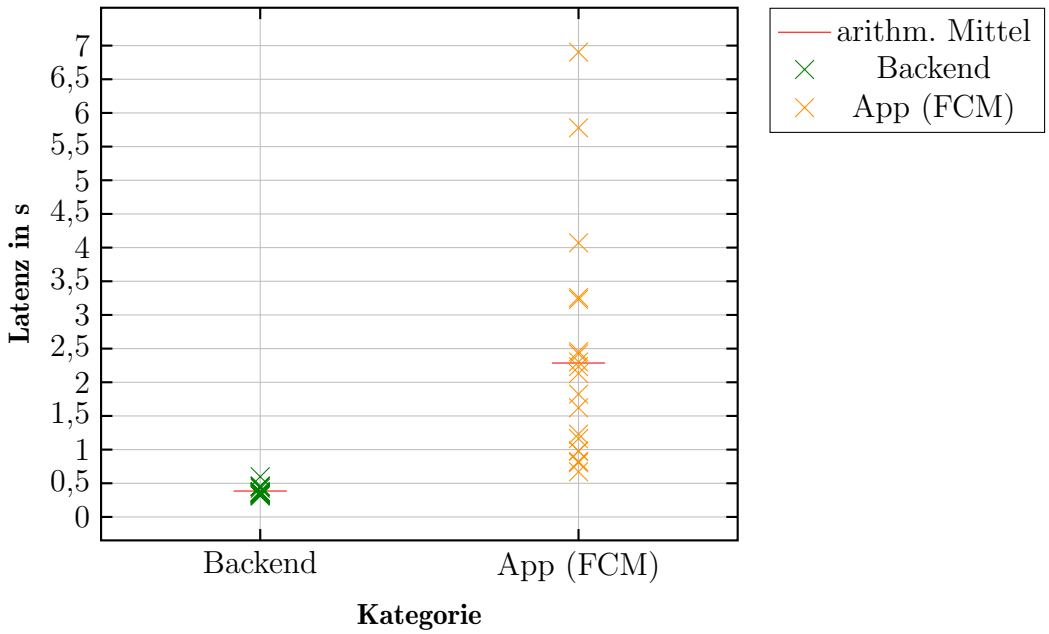

Abbildung 6.5: Verteilung der Latenzzeiten bei schlechter Netzwerkverbindung.

Analyse und Interpretation

Die Messungen bestätigen, dass die in Anforderung NFA 3 definierten Reaktionszeiten eingehalten werden. Selbst bei schwankender Netzwerkverbindung wurden die Benachrichtigungen stets innerhalb von weniger als acht Sekunden zugestellt.

In der praktischen Anwendung ist jedoch zu berücksichtigen, dass der Sensor zusätzliche Zeit benötigen kann, um die Daten an ioBroker zu übermitteln. Dennoch sollte dieser Prozess innerhalb der in den Anforderungen festgelegten Zehn-Sekunden-Grenze (NFA 3b) bleiben.

Die Ergebnisse verdeutlichen, dass die Netzwerkverbindung des Empfangsgeräts einen entscheidenden Einfluss auf die Reaktionszeiten hat. Dies ist insbesondere für zeitkritische Benachrichtigungen relevant.

6.2.2 Durchsatz

Der Durchsatz des Benachrichtigungssystems ist von zentraler Bedeutung, da er maßgeblich zur Erfüllung der Anforderungen an die Zuverlässigkeit (NFA 1) und Skalierbarkeit (NFA 2) beiträgt. Im Folgenden wird untersucht, ob das entwickelte System in der Lage ist, eine große Anzahl von Benachrichtigungen innerhalb eines kurzen Zeitraums zuverlässig zu verarbeiten.

Zur Messung des Durchsatzes wurde das Test-Setup gemäß der Beschreibung in Kapitel 5.3 Implementierung des Systems Seite 57 aufgebaut.

Messmethodik und Einflussfaktoren

Um den Durchsatz des Systems zu bewerten, wurde eine Reihe von Tests durchgeführt, bei denen große Mengen an Benachrichtigungen innerhalb kurzer Zeiträume versendet wurden. Ziel dieser Untersuchungen war es, das Verhalten des Systems unter hoher Last zu analysieren und zu prüfen, inwieweit eine zuverlässige und zeitnahe Zustellung der Nachrichten gewährleistet werden kann. Zusätzlich wurde beobachtet, ob die Nachrichten in der ursprünglichen Reihenfolge beim Empfänger eintreffen. Außerdem wurden Tests durchgeführt, bei denen Benachrichtigungen aus mehreren Instanzen des Adapters in ioBroker simultan gesendet wurden.

Messungen

Insgesamt wurden 15 Messungen durchgeführt, bei denen jeweils zwischen zehn und 100 Nachrichten pro Durchlauf versendet wurden. Die Übertragung der jeweiligen Nachrichtenmengen erfolgte mithilfe eines Skripts (siehe Listing A.4, S. 107) innerhalb eines Zeitraums von bis zu 0,181 s. Die Sendezeit pro Nachricht lag in der Regel innerhalb einer Sekunde, was als akzeptabler Wert bewertet werden kann. Ein Zusammenhang zwischen der Anzahl versendeter Nachrichten und der jeweiligen Sendezeit ließ sich nicht feststellen, was darauf hindeutet, dass das System auch bei

hoher Last zuverlässig arbeitet.

Auffällig ist jedoch, dass bei den ersten vier Messungen sehr hohe Fehlerraten von 40 bis 60 Prozent auftraten (siehe Tab 6.1). Auch in einer späteren Messung kam eine Nachricht nicht an. Die Fehlerrate verbesserte sich zwar ab der fünften Messung signifikant, obwohl der Zeitabstand zwischen der vierten und fünften Messung nur fünf Minuten betrug, während die anderen Messungen deutlich größere Zeiträume dazwischen hatten. Die verloren gegangenen Nachrichten wurden zwar im Backend gespeichert, sodass sie nach einer Aktualisierung in der App angezeigt werden konnten. Jedoch wurden sie nicht als Push-Benachrichtigung über FCM zugestellt, da hier ein Fehler auftrat. Dieser Fehler liegt im Backend und resultierte daraus, dass innerhalb eines kurzen Zeitraums mehrere Anfragen zur Initialisierung der Firebase-App gestellt wurden, wodurch versucht wurde, eine neue Instanz der Firebase App zu erstellen, obwohl bereits eine existierte. Der Ursprung dieses Problems ist, dass das Backend zuvor die Anfragen zum Versand von Push-Benachrichtigungen seriell bearbeitete. Die Bearbeitung der Benachrichtigungen für Nutzer erfolgt jedoch parallel. Die parallele Abarbeitung führt nicht nur zum beschriebenen Fehler, sondern auch dazu, dass Benachrichtigungen nicht in der richtigen Reihenfolge bei dem Nutzer ankommen.

Tabelle 6.1: Messungen des Durchsatzes mit Fehlerraten und Sendezeiten pro Nachricht in s.

Nr.	**Gesamtanzahl der Nachrichten**	**Angekommene Nachrichten**	**Fehlerrate**	**Sendezeit pro Nachricht (s)**
1.	10	6	40 %	0,343
2.	10	5	50 %	1,097
3.	10	5	50 %	0,689
4.	10	4	60 %	0,634
5.	10	10	0 %	0,233
6.	10	10	0 %	n.g.
7.	10	10	0 %	n.g.
8.	10	10	0 %	n.g.
9.	20	20	0 %	n.g.
10.	20	20	0 %	n.g.
11.	20	20	0 %	n.g.
12.	50	50	0 %	n.g.
13.	100	99	1 %	n.g.
14.	100	100	0 %	0,097
15.	100	100	0 %	0,106

Analyse und Interpretation

Hinsichtlich des Durchsatzes (NFA 2) zeigt das System Optimierungspotenzial. Die derzeitige Architektur des Backends könnte bei einer erheblichen Zunahme der Nutzer oder Nachrichten zu Performanceproblemen führen.

Es wurde nicht untersucht, inwieweit sich die gleichzeitige Nutzung durch mehrere Benutzer auf den Durchsatz des Systems auswirkt. Basierend auf der Systemarchitektur ist jedoch davon auszugehen, dass der Durchsatz weitgehend unverändert bleibt, da der limitierende Faktor das Push-Backend darstellt. ioBroker sendet alle Benachrichtigungen an denselben Endpunkt, unabhängig davon, ob die Nachrichten von einem einzelnen oder von mehreren Nutzern stammen.

6.3 Vergleich mit bestehenden Lösungen

Wie bereits in der Tabelle 4.2 zum Vergleich der bestehenden Benachrichtigungssysteme Seite 33 aufgezeigt, erfüllt keines der untersuchten Systeme alle gestellten Anforderungen. Insbesondere im Bereich der funktionalen Anforderungen ergaben sich Defizite, da die bestehenden Lösungen nicht vollständig auf den spezifischen Anwendungsfall abgestimmt sind.

Wesentliche Funktionen, wie der Abruf des Benachrichtigungsverlaufes oder die Verwaltung von Nutzern und Geräten, waren nicht in allen Benachrichtigungssystemen verfügbar. Darüber hinaus ermöglichten lediglich Pushover und Gotify die Integration in eigene Anwendungen mit dem Empfang von Push-Benachrichtigungen. Daher wurde bei der Entwicklung des generischen Benachrichtigungssystems besonderer Wert darauf gelegt, diese Anforderungen gezielt zu berücksichtigen, jedoch gibt es für dieses System noch kein SDK, was die Einbindung erleichtern würde.

Auch hinsichtlich der nichtfunktionalen Anforderungen wurden bei den bestehenden Systemen signifikante Einschränkungen festgestellt. Mit Ausnahme von Gotify zeigte sich bei den anderen Systemen eine starke Skalierung der Kosten bei einer hohen Anzahl versendeter Push-Benachrichtigungen. Das neu implementierte System ist von diesem Problem jedoch nicht betroffen. Allerdings steigen bei höherer Auslastung die Kosten für das Hosting auf Azure, da in solchen Fällen die Ressourcen des PushBackends aufgestockt werden müssen.
Zudem ergaben sich bei den bestehenden Systemen Datenschutzrisiken, da die Speicherung und Verarbeitung der Daten auf den Servern der Anbieter in verschiedenen Ländern erfolgt und somit nicht vollständig unter eigener Kontrolle liegt.
Jedoch liegen die Schwächen des neuen Systems im Durchsatz, insbesondere bei einer hohen Anzahl an gleichzeitigen Push-Benachrichtigungen. Allerdings wurden die bestehenden Systeme in diesem Bereich nicht getestet, sodass die Aussagen nur auf den Informationen basieren, die auf der Website des Anbieters zur Verfügung stehen.

Wie in der Tabelle 6.2 Seite 75 dargestellt, erfüllt das neu implementierte System alle definierten Anforderungen ausreichend mit Optimierungspotenzial.

Tabelle 6.2: Überprüfung der Anforderungen des implementierten Systems.

Anforderung	**ioBroker mit Adapter und Sensor**	**Develappers PushBack-end**	**FCM**	**Ges.**
Ereignisse erkennen und Aktionen ausführen (FA 1a)	✓	n.r.	n.r.	✓
Push-Benachrichtigung senden (FA 1b)	n.r.	✓	✓	✓
Benachrichtigung mit Titel, Text und Zeitstempel (FA 1c)	✓	✓	n.r.	✓
Speicherung der Benachrichtigungen (FA 2)	n.r.	✓	n.r.	✓
Mandantenfähigkeit (FA 3)	✓	✓	n.r.	✓
Geräteverwaltung (FA 4)	n.r.	✓	n.r.	✓
Integration in Anwendungen (FA 5)	n.r.	✓	(✓) noch kein offizielles SDK	(✓)
Zuverlässigkeit (NFA 1)	✓	✓	✓	✓
Skalierbarkeit (NFA 2)	✓	(✓)	✓	(✓)
Reaktionszeiten (NFA 3)	✓	✓	✓	✓
Wartbarkeit und Erweiterbarkeit (NFA 4)	✓	✓	✓	✓
Ressourcennutzung (NFA 5)	✓	✓	n.r. (Cloud)	✓
Aktualität und Zukunftsfähigkeit (NFA 6)	(✓)	✓	✓	✓
Externe Dienste (NFA 7)	ggf. Hosting-Gebühren	Hosting-Gebühren	✓	✓
Datenschutz (NFA 8)	✓	✓	✓	✓
Benutzerfreundlichkeit (NFA 9)	✓	n.r.	n.r.	✓
Kompatibilität (NFA 10)	✓	n.r.	n.r.	✓

Anforderung	ioBroker mit Adapter und Sensor	Develappers PushBack-end	FCM	Ges.
Portabilität (NFA 11)	n.r.	n.r.	✓	✓

6.4 Potenzial für zukünftige Verbesserungen

Das aktuelle Benachrichtigungssystemen zeigt in Bezug auf den Durchsatz Optimierungspotenzial. Bei einer höheren Anzahl gleichzeitiger Anfragen und Nutzer treten Performanceprobleme auf, die die Effizienz des Systems beeinträchtigen. Um den Durchsatz zu verbessern, könnte die Architektur so angepasst werden, dass sie besser mit großen Datenmengen und steigenden Nutzerzahlen umgehen kann. Dies erfordert eine Änderung am Backend, um sicherzustellen, dass die Firebase-App nicht bei jeder Anfrage eine neue Instanz erstellt.

Mit dem aktuellen System kann eine zeitkritische Zustellung, abhängig von den jeweiligen Anforderungen, schwierig werden. Selbst unter optimalen Netzwerkbedingungen liegt die Zustellzeit einer Push-Benachrichtigung bei bis zu zwei Sekunden, wobei dieser Wert maßgeblich durch FCM bestimmt wird und sich nicht weiter beschleunigen lässt. Außerdem wurde die Verzögerung zwischen dem Sensor und der anschließenden Verarbeitung im ioBroker in diesem Zusammenhang noch nicht berücksichtigt.

Darüber hinaus wäre es sinnvoll, dem Nutzer erweiterte Einstellungsmöglichkeiten in ioBroker zu bieten, wie etwa die Auswahl von Klingeltönen oder die Festlegung der Priorität von Benachrichtigungen. Diese Funktionen sind bereits in anderen Systemen wie Pushover integriert (siehe Abb. A.4, S. 108) und könnten die Benutzererfahrung verbessern. Darüber hinaus ist es sinnvoll, eine Konfigurationsmöglichkeit bereitzustellen, mit der Nutzer gezielt festlegen können, an welche ihrer Endgeräte einzelne Benachrichtigungen übermittelt werden. Ergänzend wäre eine Priorisierung von Benachrichtigungen hilfreich, um sicherzustellen, dass sicherheitsrelevante Meldungen wie etwa von einem Feuermelder nicht im Nachrichtenfluss weniger wichtiger Ereignisse untergehen.

7 Benachrichtigungssystem ohne Abhängigkeit von Drittanbietern

In diesem Kapitel wird die Frage untersucht: „Kann man kritische Benachrichtigungen ohne die Abhängigkeit von Drittanbietern zustellen?“ Dabei steht die Suche nach technischen Lösungsansätzen im Fokus, die eine zuverlässige und datenschutzfreundliche Zustellung ermöglichen, unabhängig von extern betriebenen Infrastrukturen.

Kritische Benachrichtigungen zeichnen sich dadurch aus, dass sie besondere Anforderungen an Zuverlässigkeit, Sicherheit oder Datenschutz stellen. Sie können datenschutz- oder gesundheitskritisch sein, wenn sie sensible Informationen enthalten, sicherheitskritisch, wenn ihre Zustellung unbedingt erforderlich ist, unabhängig vom genauen Zeitpunkt, oder zeitkritisch.
„Abhängigkeit von Drittanbietern“ beschreibt hierbei die Situation, in der für das Benachrichtigungssystem externe Infrastrukturen genutzt werden müssen, die außerhalb der eigenen Kontrolle liegen. Diese Abhängigkeit kann problematisch sein, da sie mit Risiken wie dem Zugriff auf sensible Nutzerdaten, potenziellen Sicherheitslücken durch unzureichend gewartete Dienste und einer generellen Abhängigkeit von der Verfügbarkeit und Zuverlässigkeit externer Cloud-Infrastrukturen einhergeht. Ziel dieses Kapitel ist es daher, Lösungsansätze zu untersuchen, die die Zustellung kritischer Benachrichtigungen ermöglichen, ohne dabei auf fremde Dienste angewiesen zu sein, um so maximale Kontrolle über Datenschutz, Sicherheit und Systemstabilität zu gewährleisten.

7.1 Systemarchitektur

Die Systemarchitektur bildet die Grundlage für die Entwicklung eines Benachrichtigungssystems, das ohne Abhängigkeit von Drittanbietern auskommt. In diesem Abschnitt wird zunächst der aktuelle Stand der Technik dargestellt, um bestehende Strukturen und ihre Limitierungen aufzuzeigen. Anschließend werden alternative Technologien betrachtet, die als Basis für eine unabhängige und datenschutzfreundliche Lösung dienen könnten.

7.1.1 Stand der Technik

Der aktuelle Stand der Technik im Bereich mobiler Push-Benachrichtigungen ist stark von verschiedenen Abhängigkeiten geprägt.

Integrationsplattform

Zunächst sind die Systeme von der verwendeten Integrationsplattform abhängig, wie zum Beispiel ioBroker, HomeAssistant oder OpenHAB. Diese Plattformen bestimmen nicht nur die Sicherheitsaspekte, sondern auch, welche Sensoren und Geräte integriert werden können, was die Flexibilität bei der Auslösung von Benachrichtigungen einschränkt. Jedoch besteht bei allen drei Plattformen die Möglichkeit, eigene Plugins zu entwickeln und so die Unterstützung weiterer Geräte flexibel zu erweitern.

Push-Backend

Darüber hinaus kann eine zusätzliche Abhängigkeit vom jeweiligen Cloud-Anbieter entstehen, sofern die Benachrichtigungssysteme auf cloudbasierte Backend-Architekturen angewiesen sind. Diese zusätzliche Abhängigkeit beeinflusst die Kontrolle über Datenschutz, Verfügbarkeit und Sicherheit.

Push-Dienste

Zudem sind die bestehenden Dienste wie Pushover, Pushbullet, Gotify oder Pushsafer für die eigentliche Zustellung von Benachrichtigungen auf die grundlegenden Zustelldienste wie Apple Push Notification Service und Firebase Cloud Messaging angewiesen.

Eine vollständig unabhängige und durchgängig kontrollierbare Zustellung kritischer Benachrichtigungen ist mit bestehenden Systemen daher nicht gewährleistet.

7.1.2 Alternative Technologien

Um ein Benachrichtigungssystem ohne Abhängigkeit von Drittanbietern umzusetzen, ist die Auswahl geeigneter Technologien von zentraler Bedeutung. In diesem Abschnitt werden alternative Ansätze vorgestellt, die das Potenzial haben, bestehende, drittanbieterabhängige Komponenten zu ersetzen. Ziel ist es, Lösungen zu identifizieren, die eine zuverlässige Kommunikation ermöglichen, ohne auf proprietäre Infrastrukturen angewiesen zu sein.

Integrationsplattform

Da es sich bei Integrationsplattformen wie ioBroker, openHAB und Home Assistant um Open-Source-Software handelt, können sie flexibel an individuelle Anforderungen angepasst werden. Dies ermöglicht eine weitgehende Unabhängigkeit von den jeweiligen Anbietern. Eine mögliche Alternative zur Nutzung bestehender Integrationsplattformen wäre der Einsatz eines Mikrokontrollers, wie zum Beispiel des ESP32, um die Steuerung und Verwaltung der Benachrichtigungen direkt zu übernehmen. Der ESP32 könnte dabei die Kommunikation zwischen dem Sensor und dem Backend übernehmen. Allerdings würde dies einen erheblichen Entwicklungsaufwand

erfordern. Sämtliche Funktionen, die eine Integrationsplattform bietet, müssten von Grund auf neu implementiert werden. Dazu gehören unter anderem die Benutzeroberfläche zur Verwaltung von Geräten, die Integration unterschiedlicher Sensoren, die Automatisierung von Abläufen und die Verwaltung von Sicherheitsaspekten.

Push-Backend

Da das Develappers PushBackend bereits als firmeninternes Produkt vorliegt, besteht hier keine Notwendigkeit zur Ablösung. Eine bestehende Abhängigkeit ergibt sich jedoch weiterhin durch das Hosting über die Cloud-Plattform Azure. Ein alternativer Ansatz für das Hosting des PushBackends wäre, dieses auf einer eigenen Infrastruktur zu betreiben. Der Vorteil einer solchen Lösung liegt in der verringerten Abhängigkeit von externen Anbietern. Ein Beispiel für die Hardware könnte ein eigener Server sein, der mit einem Windows Server 2019/2022 oder Windows 10/11 Pro betrieben wird. Für die Datenbankverwaltung könnte der Microsoft SQL Server Express oder eine höherwertige MS SQL Server Version eingesetzt werden. Die ASP.NET Core App könnte über den IIS bereitgestellt und die entsprechende App direkt dort deployed werden. Das Netzwerk müsste so konfiguriert werden, dass ein Zugriff von außen über Portfreigaben im Router ermöglicht wird. Die vollständige Kontrolle über die Infrastruktur bietet Vorteile hinsichtlich der Unabhängigkeit von Cloud-Anbietern, erfordert jedoch gleichzeitig die Verantwortung für die Implementierung von Sicherheitsmaßnahmen und die datenschutzkonforme Speicherung der Daten. Zudem können zusätzliche Wartungsaufwände entstehen, die bei einer Cloud-Lösung in der Regel vom Anbieter übernommen werden.

Push-Dienste

Eine Alternative zu FCM und APNs wäre der Einsatz von HTTP Polling oder Long Polling. Bei diesen Verfahren fragt der Client regelmäßig nach neuen Benachrichtigungen über einen langen Zeitraum (Lv u. a. 2024). Obwohl diese Methode eine einfache Möglichkeit bietet, Benachrichtigungen zu erhalten, handelt es sich nicht um ein echtes Push-Verfahren. Der Client muss aktiv nach den Benachrichtigungen fragen, anstatt benachrichtigt zu werden, sobald neue Daten verfügbar sind. Dadurch enteht auch ein hoher Ressourcenverbrauch.

Eine weitere Möglichkeit wäre, Push-Benachrichtigungen über eine eigene Lösung mit Peer-to-Peer-Kommunikation zu realisieren. Hierbei kommunizieren die Geräte direkt miteinander, ohne einen zentralen Server zu benötigen. Diese Lösung setzt jedoch voraus, dass die Geräte konstant miteinander verbunden sind oder sich innerhalb eines lokalen Netzwerks befinden. In Szenarien, in denen Geräte mobil sind oder sich nur sporadisch miteinander verbinden, ist diese Methode wenig praktikabel.

WebSocket-basierte Push-Benachrichtigungen stellen eine weitere Alternative dar. WebSockets ermöglichen eine bidirektionale, Vollduplex-Verbindung (Zhang u. a. 2013), die eine dauerhafte Kommunikation zwischen Client und Server ermöglicht. Diese Technik sorgt für geringe Verzögerungszeiten und schnelle Reaktionszeiten, was sie besonders für Echtzeitanwendungen geeignet macht (Eka Putra u. a. 2024).

WebSockets nutzen bestehende TCP-Verbindungen und reduzieren den Kommunikations-Overhead im Vergleich zu HTTP Polling, da sie kleinere Header verwenden (Łasocha u. a. 2021). Diese dauerhafte Verbindung kann jedoch zu einem hohen Stromverbrauch bei mobilen Geräten führen, da die App ständig mit dem Server verbunden bleiben muss (Eka Putra u. a. 2024). Wenn die Verbindung zwischen dem Client und dem Server unterbrochen wird, kann der Server benachrichtigt werden, was eine Reaktion ermöglicht (Eka Putra u. a. 2024), etwa die Speicherung der Daten bis zur Wiederherstellung der Verbindung. Allerdings kann es bei Apps, die im Hintergrund laufen zu Problemen bei der zuverlässigen Zustellung von Benachrichtigungen kommen. Dies resultiert daraus, dass Betriebssysteme mobiler Geräte die Hintergrundaktivitäten von Apps stark begrenzen, um den Ressourcenverbrauch zu minimieren. Insbesondere auf iOS werden Apps im Hintergrund stark begrenzt, was es unmöglich macht, Push-Benachrichtigungen über WebSockets zuzustellen (Apple Engineer 2021). Ein Beispiel für die Nutzung von WebSockets für Push-Benachrichtigungen ist Gotify, jedoch ist diese Lösung nur für Android verfügbar (Gotify 2025a).

Eine weitere Möglichkeit zur Zustellung von Push-Benachrichtigungen ist MQTT, ein leichtgewichtiges Messaging-Protokoll, das speziell für das Internet der Dinge entwickelt wurde (Eka Putra u. a. 2024). MQTT ermöglicht eine effiziente Kommunikation zwischen Geräten in verteilten Netzwerken (Eka Putra u. a. 2024). Es nutzt ein Publish/Subscribe-Verfahren, bei dem ein Broker die Nachrichten verwaltet (siehe Abb. 7.1), wodurch die Kommunikation skalierbar und ressourcenschonend wird (Tang u. a. 2013). MQTT zeichnet sich durch geringe Bandbreitenanforderungen, Effizienz und stromsparende Eigenschaften aus (Eka Putra u. a. 2024). Zudem ermöglicht es eine hohe Skalierbarkeit, sodass es für Millionen von Geräten eingesetzt werden kann (MQTT 2024). Es liegt unter anderem ein Artikel von Tang vor, der zeigt, dass es möglich ist, ein Push-Benachrichtigungssystem auf Basis von MQTT zu entwickeln (vgl. Tang u. a. 2013). Außerdem existiert mit Pushy ein Dienst, der für den Empfang von Nachrichten das MQTT-Protokoll verwendet (Pushy 2023).

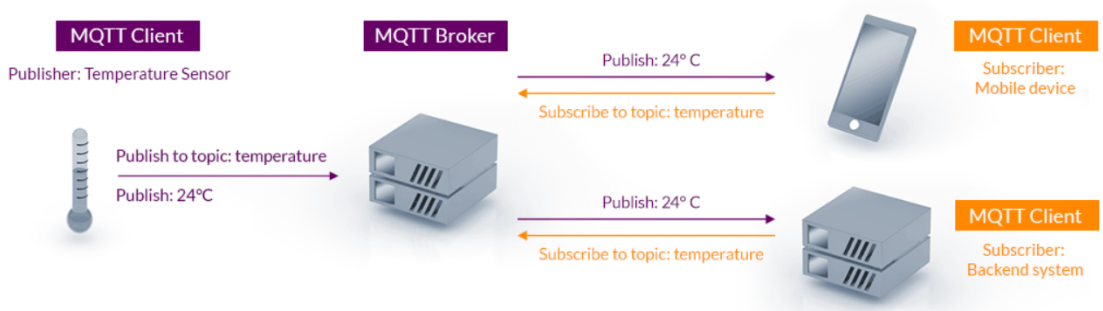

Abbildung 7.1: MQTT Architektur. (Quelle: MQTT 2024)

Ein direkter Vergleich zwischen WebSockets und MQTT, durchgeführt von Eka Putra, zeigt, dass WebSockets hinsichtlich der durchschnittlichen End-to-End-Latenz mit 0,7 s effizienter sind als MQTT, welches eine durchschnittliche Verzögerung von 1,008 s aufweist (Eka Putra u. a. 2024). Dennoch bietet MQTT entscheidende Vorteile im Hinblick auf Zuverlässigkeit sowie eine effizientere Ressourcennutzung, was

insbesondere für den Einsatz in verteilten Netzwerken und auf ressourcenbeschränkten Endgeräten von Bedeutung ist (Eka Putra u. a. 2024). Beide Technologien stehen jedoch vor Herausforderungen, insbesondere in Bezug auf die Zustellung von Push-Benachrichtigungen im Hintergrundbetrieb. Unter Android sind Einschränkungen bezüglich der Hintergrundverarbeitung zu beachten, welche die Zuverlässigkeit der Benachrichtigungszustellung beeinträchtigen können. Deshalb nutzt Pushy in der eigenen Demo App auch einen FCM Fallback (Pushy 2024 Zeile 65). Unter iOS gestaltet sich die Situation noch restriktiver: Aufgrund der strikten Begrenzung der Hintergrundaktivitäten durch das Betriebssystem können Anwendungen im Hintergrund keine dauerhafte Verbindung aufrechterhalten (Apple Engineer 2021), was die Nutzung von MQTT oder WebSockets für Push-Benachrichtigungen ausschließt. Lediglich in geschlossenen Netzwerken, in denen eine direkte Verbindung zwischen Geräten besteht, könnte auf APNs verzichtet werden (Sawant 2020).

Im Kontext des vorliegenden Anwendungsszenarios stellt MQTT dennoch die geeignetste Lösung dar, da es aufgrund seines geringen Ressourcenverbrauchs besonders für mobile Endgeräte mit beschränkten Leistungsreserven vorteilhaft ist.

Systemzusammenstellung ohne Drittanbieter

Ein alternatives System zur Zustellung von Push-Benachrichtigungen könnte die Integration eines Sensors sein, beispielsweise einem Magnetsensors, der den Zustand eines Fensters überwacht. Der Sensor könnte über einen ESP32 Mikrocontroller über HTTPS mit dem Develappers PushBackend, kommunizieren. Anschließend würde die Benachrichtigung durch eine Kommunikationstechnologie wie MQTT beziehungsweise APNs an die Individualanwendung übertragen werden (siehe Abb. 7.2).

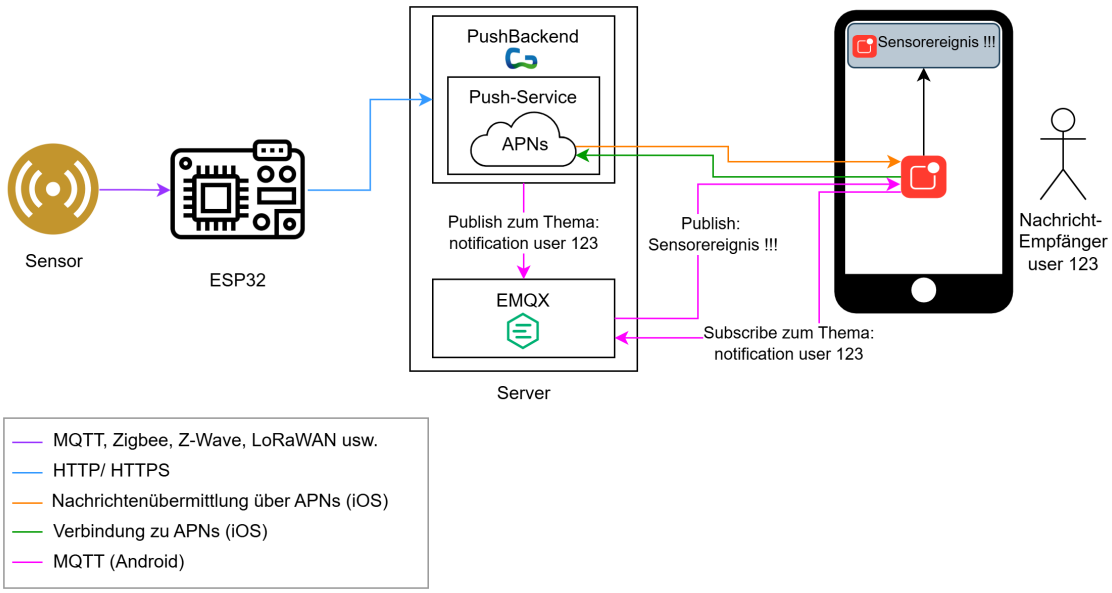

Abbildung 7.2: Systemarchitektur für ein generisches Benachrichtigungssystem mit wenigen Drittanbietern.

7.1.3 Herausforderungen und Einschränkungen

Eine Umsetzung eines Benachrichtigungssystems ohne die Abhängigkeit von Drittanbietern ist grundsätzlich für Android-Geräte möglich, da auf diesem Betriebssystem alternative Zustellmethoden wie WebSockets oder MQTT genutzt werden können. Für iOS hingegen ist eine vollständige Unabhängigkeit von Drittanbietern aufgrund der zwingenden Notwendigkeit von Apple Push Notification Service nicht realisierbar, da iOS sonst keine zuverlässige Benachrichtigungszustellung zulässt. Zudem erfordert die Implementierung einer eigenen Infrastruktur einen erheblichen Entwicklungsaufwand, da alle Funktionen einer Integrationsplattform von Grund auf neu entwickelt werden müssten. Aufgrund des hohen Aufwands und der praktischen Einschränkungen ist eine vollständige Realisierung eines unabhängigen Benachrichtigungssystems im Rahmen dieser Arbeit nicht umsetzbar.

7.2 Leistungsbewertung der MQTT-Kommunikation

Um die MQTT-basierte Kommunikationsverbindung (siehe Abb. 7.3) als alternative Lösung zu FCM unter Android zu testen, wurde eine MQTT Client App auf Basis des plattformübergreifenden Frameworks .NET MAUI entwickelt und implementiert. Die Anwendung ist auf das MQTT-Topic `mqtt_push_user123` abonniert und reagiert auf eingehende Nachrichten mit dem Auslösen einer Push-Benachrichtigung. Alle empfangenen Nachrichten werden zusätzlich in einer Verlaufsliste innerhalb der App dargestellt (siehe Abb. A.5, S. 109).

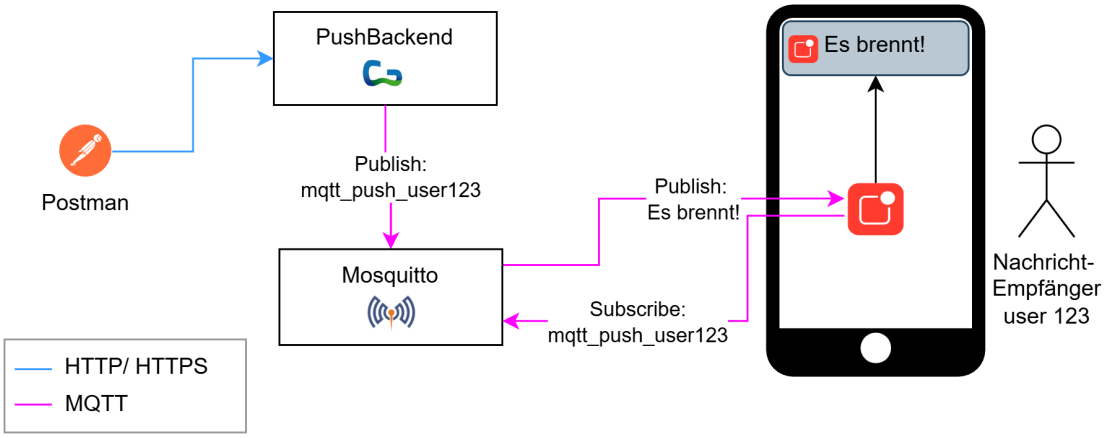

Abbildung 7.3: Aufbau des Tests der MQTT-Verbindung.

Damit die App auch im minimierten oder geschlossenen Zustand zuverlässig auf MQTT-Nachrichten reagieren kann, musste in den App-Einstellungen die Option Hintergrundnutzung uneingeschränkt zulassen aktiviert werden (siehe Abb. A.6, S. 109). Diese Funktionalität ist entscheidend, da Hintergrundprozesse standardmäßig restriktiv behandelt werden. Außerdem musste ein Foreground Service (ANDROID

Developer 2025) eingerichetet werden, der dazu dient die MQTT-Verbindung aufrecht zu halten.

Auf Serverseite wurde das MQTT-Protokoll in das entwickelte Push-Backend integriert. Über einen Publish-Vorgang auf das Topic `mqtt_push_user123` kann eine Nachricht versendet und damit gezielt eine Benachrichtigung auf dem Endgerät des Nutzers ausgelöst werden. Für Testzwecke erfolgte das Auslösen der Benachrichtigung manuell über das Tool Postman und das PushBackend wurde wieder über Azure gehostet. Als MQTT-Broker kam die öffentlich zugängliche Testinstanz `tcp://test.mosquitto.org:1883` von Mosquitto zum Einsatz.

Die Leistungsfähigkeit eines Systems wird maßgeblich durch Faktoren wie Zuverlässigkeit, Latenz und Durchsatz bestimmt. Eine geringe Latenz ist für schnelle Informationsverarbeitung entscheidend, während eine guter Durchsatz das System bei hoher Last stabil hält.

Das System zur Leistungsfestellung von MQTT wurde gemäß Abbildung 7.3 Seite 82 aufgebaut. Die Messmethoden des Durchsatzes und die Einflussfaktoren entsprechen denen, die in Kapitel 6.2 Leistungsanalyse Seite 67 beschrieben wurden. Zusätzlich hat der gewählte MQTTBroker Einfluss auf das System.

7.2.1 Zuverlässigkeit

Zur Untersuchung der Zuverlässigkeit (NFA 1) wurden vier Testgeräte über einen Zeitraum von 21,67 Stunden beobachtet (siehe Tab A.5, S. 112). Die Geräte liefen unter den Android-Versionen 13 bis 15, wobei die MQTT-Client-App auf allen Geräten vollständig geschlossen war. Drei der Geräte (Tab 7.1 Nr. 1 bis 3) befanden sich dabei am selben Standort und wurden während des Tests nicht aktiv genutzt.

Tabelle 7.1: Verbindungsabbrüche bei verschiedenen Geräten mit MQTT Verbindung über einen Zeitraum von 21,67 h

Nr.	**Testgerät**	**Anz.**	**Zeit**				
			min	**max**	**Ø**	**Gesamt**	**Offline**
1	Samsung A14	6	00:04:42	00:37:14	00:17:25	02:01:57	9,4 %
2	Samsung A51	0	0	0	0	0	0 %
3	Google Pixel 7a	2	00:01:44	00:07:46	00:04:45	00:09:30	0,7 %
4	Google Pixel 7a	0	0	0	0	0	0 %

Trotz identischer Rahmenbedingungen zeigte sich ein unterschiedliches Verhalten hinsichtlich der Verbindungsstabilität: Zwar empfingen alle Geräte die insgesamt neun versendeten Nachrichten korrekt, bei zwei Geräten kam es jedoch zu mehrfachen temporären Verbindungsabbrüchen (siehe Tab 7.1 Nr. 1 und 3). Das Gerät war somit 9,4 % der Zeit nicht mit dem Broker verbunden. Auffällig war, dass sich das Gerät jeweils automatisch nach spätestens nach rund 38 Minuten wieder mit dem Broker verband.

Die Ursachen für diese Unterschiede konnten nicht eindeutig identifiziert werden, zumal selbst zwei baugleiche Geräte unterschiedliches Verhalten zeigten. Jedoch konnte ausgeschlossen werden, dass es an der Internetverbindung oder dem MQTT-Broker liegt. Interessanterweise blieb das aktiv genutzte Pixel 7a durchgehend verbunden, obwohl es sich im Gegensatz zu den anderen Geräten nicht konstant am gleichen Ort befand.

7.2.2 Latenzzeiten

Insgesamt wurden 40 Messungen durchgeführt (siehe Tab A.6, S. 114). Erfasst wurde dabei die Zeitspanne zwischen dem Versand der Nachricht durch das Backend an den MQTT-Broker und der Anzeige der Benachrichtigung in der App (siehe Abb. 7.4).

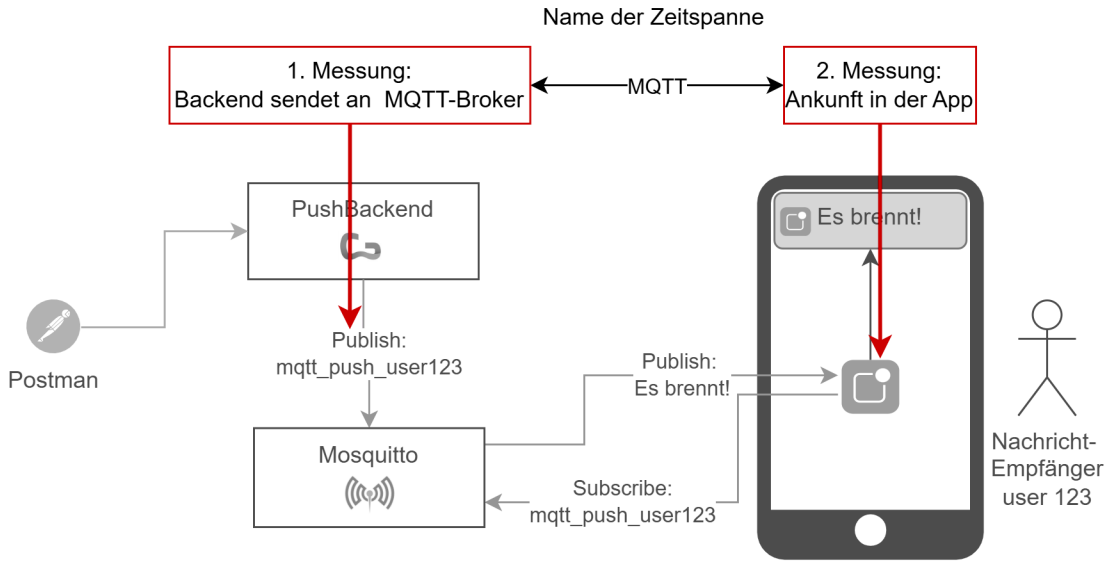

Abbildung 7.4: Messpunkte zur Bestimmung der Latenzzeiten der MQTT Verbindung.

Die ersten 20 Messungen fanden unter optimalen Netzwerkbedingungen statt. Das Empfangsgerät war hierbei mit einem 5G-Mobilfunknetz verbunden, das eine durchschnittliche Download-Geschwindigkeit von 31 Mbit/s sowie eine Upload-Geschwindigkeit von acht Mbit/s aufwies. Die gemessenen Latenzzeiten bewegten sich in dieser Stichprobe zwischen 0,095 s und 0,214 s (siehe Abb. 7.5). Das arithmetische Mittel der Zustellzeiten betrug 0,151 s, bei einer Standardabweichung von 0,029 s.

Anschließend wurden weitere 20 Messungen durchgeführt, bei denen sich das Empfangsgerät in einem 2G-Mobilfunknetz befand. Die Verbindung wies eine Download-Geschwindigkeit von 0,08 Mbit/s sowie eine Upload-Geschwindigkeit von 0,04 Mbit/s auf. Dabei zeigte sich ein deutlicher Unterschied in der Zustellzeit der Push-Benachrichtigungen in der App. Unter diesen eingeschränkten Netzwerkbedingungen

variierten die gemessenen Latenzzeiten erheblich, lagen jedoch durchweg auf einem signifikant höheren Niveau, zwischen 0,274 s und 9,181 s (siehe Abb. 7.5). Das arithmetische Mittel der Zustellzeiten betrug 2,249 s, bei einer hohen Standardabweichung von 2,149 s.

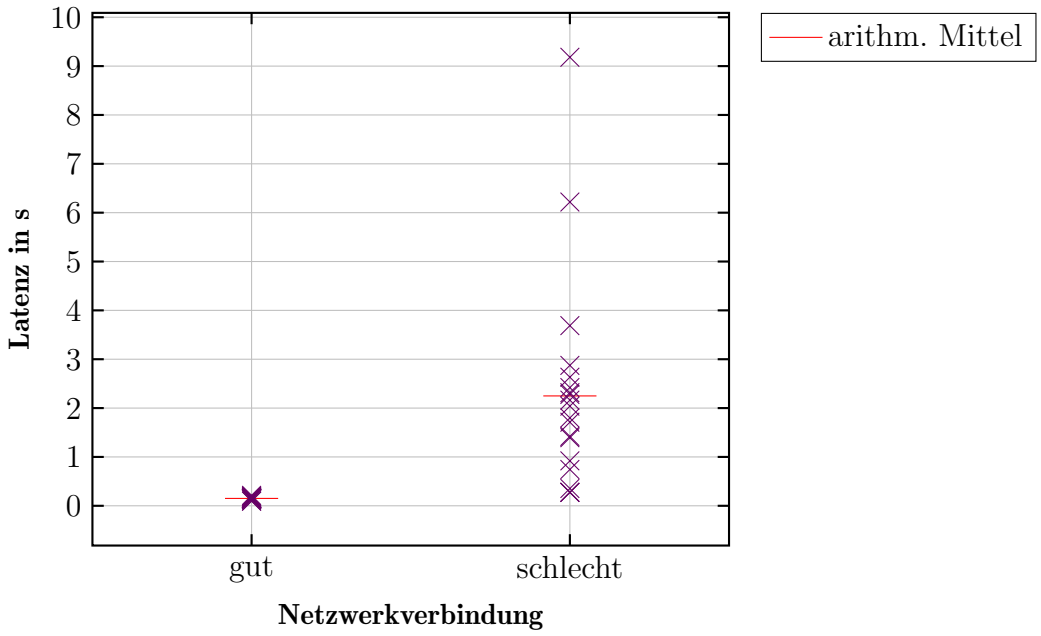

Abbildung 7.5: Verteilung der Latenzzeiten bei guter und schlechter Netzwerkverbindung unter Verwendung von MQTT.

Im Vergleich zur Zustellung über FCM zeigt MQTT bei guter Netzwerkverbindung eine deutlich niedrigere Latenz. Sie fällt um 0,474 s geringer aus (siehe Abb. 7.6, S. 86). Auch die Standardabweichung ist um 0,215 s reduziert, was auf eine schnellere Zustellung hinweist (siehe Tab 7.2, S. 87).

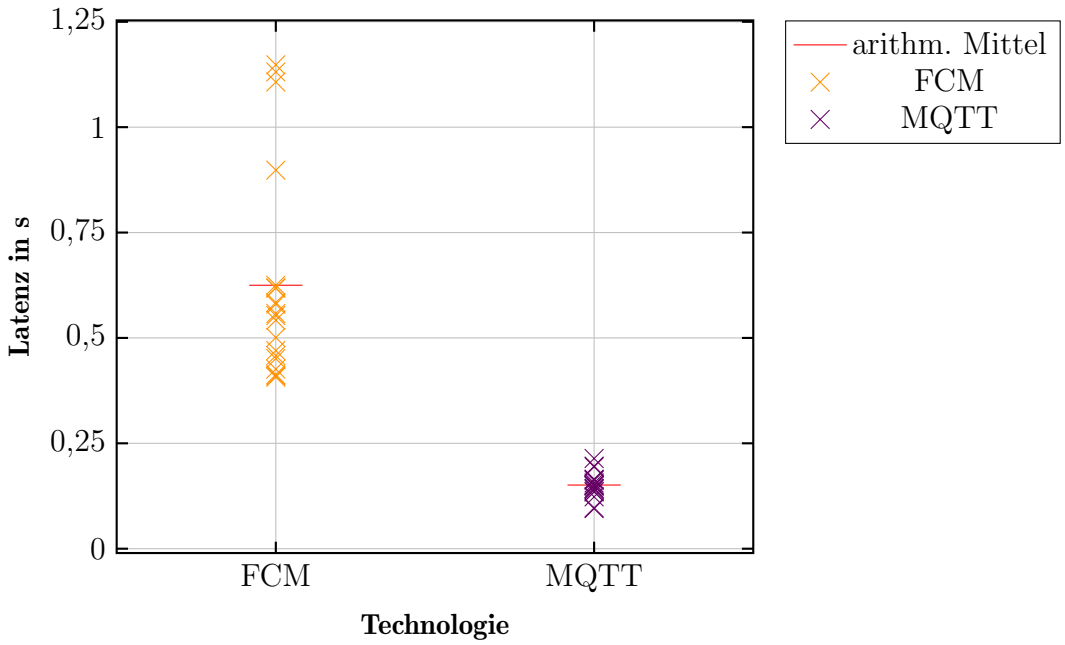

Abbildung 7.6: Vergleich der Verteilung der Latenzzeiten zwischen MQTT und FCM bei guter Netzwerkverbindung.

Jedoch zeigt MQTT Schwächen bei der Zustellung unter schlechten Netzwerkbedingungen. Zwar unterscheidet sich das arithmetische Mittel der Latenzzeiten im Vergleich zu FCM nur geringfügig, jedoch führt eine um 0,458 s höhere Standardabweichung zu vereinzelten, stark verzögerten Zustellungen von bis zu 9,181 s (siehe Abb. 7.7). Dadurch kann die Einhaltung der Zehn-Sekunden-Anforderung (NFA 3b) problematisch werden.

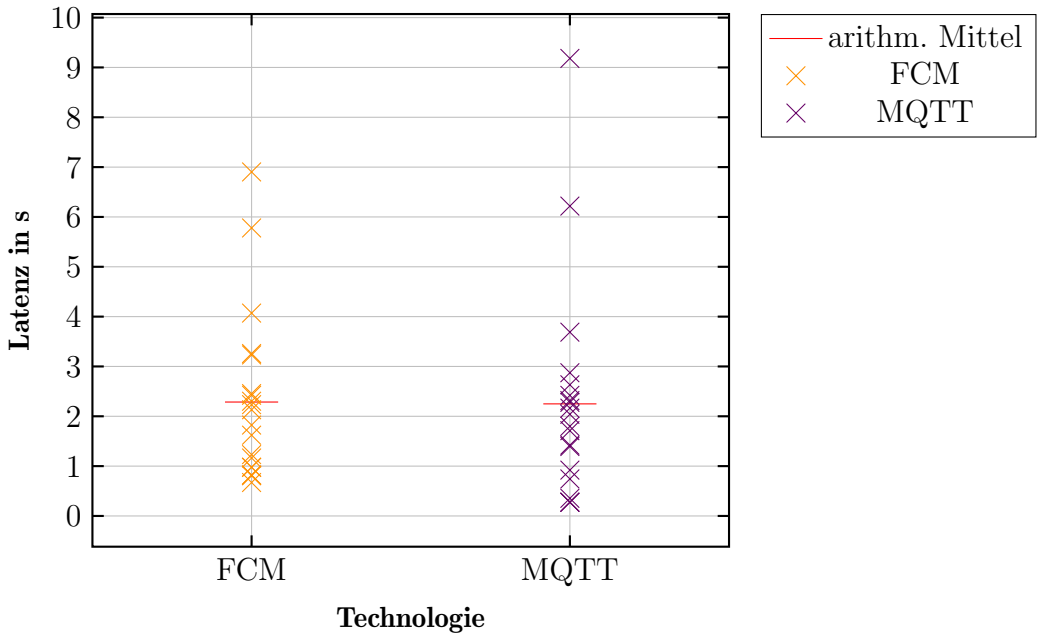

Abbildung 7.7: Vergleich der Verteilung der Latenzzeiten zwischen MQTT und FCM bei schlechter Netzwerkverbindung.

Unter guten Netzwerkbedingungen liefert MQTT Push-Benachrichtigungen schneller und stabiler als FCM. Bei schlechter Verbindung zeigt FCM eine verlässlichere Zustellung, während MQTT häufiger und deutlich stärkere Verzögerungen verursacht, obwohl sich das arithmetische Mittel der Latenzzeiten kaum unterscheidet (siehe Tab 7.2). Die Messergebnisse belegen, dass die in Anforderung NFA 3 definierten Reaktionszeiten unter stabilen Netzwerkbedingungen eingehalten werden. Unter eingeschränkten Verbindungsverhältnissen treten teilweise größere Verzögerungen auf, dennoch erfolgt die Zustellung innerhalb des geforderten Zeitrahmens von zehn Sekunden. Jedoch ist zu beachten, dass die Verzögerung vom Sensor bis zum Eintreffen der Daten im Backend nicht Bestandteil der Messung war.

Tabelle 7.2: Vergleich der Latenzzeiten zwischen FCM und MQTT unter verschiedenen Netzbedingungen (Werte in s).

Netzverbindung	Kategorien	FCM	MQTT
gut	Anz. Messungen	20	20
	min	0,407	0,095
	max	1,148	0,214
	arithm. Mittel	0,625	0,151
	Stdabw. (S)	0,244	0,029
schlecht	Anz. Messungen	20	20
	min	0,671	0,274
	max	6,903	9,181
	arithm. Mittel	2,286	2,249
	Stdabw. (S)	1,691	2,149

Die Ergebnisse verdeutlichen, dass die Netzwerkqualität des Empfangsgeräts einen wesentlichen Einfluss auf die Reaktionszeiten bei der Nutzung von MQTT hat.

7.2.3 Durchsatz

Zur Bewertung des Durchsatzes wurden Lasttests mit einer hohen Anzahl an Benachrichtigungen in kurzer Zeit durchgeführt.

Insgesamt wurden 15 Messungen durchgeführt (siehe Tab 7.3, S. 88), bei denen jeweils zwischen zehn und 200 Nachrichten pro Durchlauf versendet wurden. Das Auslösen der Benachrichtigungen erfolgte mithilfe des Postman Runners. Für die Berechnung der gesamten Sendezeit pro Nachricht wurde das zuvor ermittelte arithmetische Mittel von 0,348 s für die Übertragung vom Sensor bis zum Backend angenommen (siehe Tab A.4, S. 112) und entsprechend aufaddiert. Die gemessene Sendezeit pro Nachricht lag durchgängig unter 0,55 s und ist somit als performant einzustufen. Ein signifikanter Zusammenhang zwischen der Anzahl versendeter Nachrichten und der jeweiligen Sendezeit konnte nicht festgestellt werden.

Tabelle 7.3: Messungen des Durchsatzes mit Fehlerraten und Sendezeiten pro Nachricht mit MQTT.

Gesamtanzahl der Nachrichten	**Fehlerrate**	**Reihenfolge**	**Sendezeit pro Nachricht (s)**
10	0 %	✓	0,438
10	0 %	✓	0,430
10	0 %	✓	0,439
10	0 %	✓	0,430
50	0 %	✓	0,421
50	0 %	✓	0,424
50	0 %	✓	0,537
50	0 %	✓	0,539
100	0 %	✓	0,417
100	0 %	✓	0,428
100	0 %	✓	0,543
150	0 %	×	0,515
150	0 %	×	0,534
200	0 %	×	0,463
200	0 %	×	0,502

In allen Messungen wurden die versendeten Nachrichten vollständig zugestellt. Ab einer Anzahl von 150 Nachrichten wurde beobachtet, dass die Zustellung im Benachrichtigungsverlauf der App zwar weiterhin in der ursprünglichen Reihenfolge erfolgte, die angezeigten Push-Benachrichtigungen jedoch nicht mehr der ursprünglichen Reihenfolge entsprachen.

Im Vergleich zu den Messungen mit FCM erfolgte die Zustellung der Nachrichten über MQTT insgesamt schneller und zuverlässiger. Dabei ist jedoch zu berücksichtigen, dass die bei FCM beobachtete Fehlerrate auf Speicherprobleme im Backend zurückzuführen war. Da MQTT lediglich testweise implementiert wurde und für diese Benachrichtigungen keine Speicherung im Backend erfolgte, konnte dieser Fehler in diesem Fall nicht auftreten.

7.3 Schlussfolgerung

Die Untersuchung zeigt, dass die Entwicklung eines Benachrichtigungssystems ohne Abhängigkeit von Drittanbietern grundsätzlich für Android-Geräte realisierbar ist. Bestehende Lösungen zeigen erhebliche externe Abhängigkeiten, die Risiken für Datenschutz und Sicherheit darstellen können. Alternative Technologien wie MQTT oder WebSockets ermöglichen eine eigenständige Zustellung von Benachrichtigungen, stoßen jedoch insbesondere auf iOS-Systemen aufgrund der notwendigen Nutzung von APNs an ihre Grenzen.

Im Gegensatz zu FCM verfügen Protokolle wie MQTT nicht über eine integrierte

Mechanik zur puffergestützten Zustellung von Nachrichten im Falle unterbrochener Netzwerkverbindungen. Eine zuverlässige Nachrichtenzustellung bei temporärer Unerreichbarkeit des Endgeräts erfordert daher zusätzliche Implementierungen auf Anwendungsseite.

In Bezug auf die Zuverlässigkeit zeigt MQTT Schwächen, da es zu gelegentlichen Verbindungsabbrüchen kommen kann. Unter stabilen Netzwerkbedingungen liefert MQTT Nachrichten jedoch schneller als FCM. In instabilen Netzen erweist sich FCM hingegen als robuster und zuverlässiger. Die geforderte maximale Reaktionszeit von zehn Sekunden wird von beiden Systemen in der Regel eingehalten. Bei schwankender Verbindung können allerdings insbesondere bei der Nutzung von MQTT vereinzelt stärkere Verzögerungen auftreten.

8 Fazit und Ausblick

Dieses abschließende Kapitel fasst die wesentlichen Erkenntnisse der Arbeit zusammen und bewertet die erreichten Ergebnisse im Hinblick auf die eingangs formulierten Ziele. Neben der technischen Umsetzung werden auch die gewonnenen praktischen Erfahrungen beleuchtet. Abschließend wird ein Ausblick auf mögliche Weiterentwicklungen und offene Fragestellungen gegeben.

8.1 Zusammenfassung der Erkenntnisse

Ziel dieser Arbeit war die Konzeption und prototypische Umsetzung eines generischen Benachrichtigungssystems, das flexibel in Individualanwendungen integrierbar ist und auf sensorbasierte Ereignisse in Echtzeit reagieren kann. Zudem wurde untersucht, inwiefern ein solches System unabhängig von externen Drittanbieterdiensten realisierbar ist.

Der entwickelte Prototyp erfüllt die definierten funktionalen und nichtfunktionalen Anforderungen in hohem Maße. Das System erwies sich als stabil, zuverlässig und vielseitig einsetzbar. Besonders hervorzuheben ist die Einhaltung kurzer Reaktionszeiten, selbst bei instabilen Netzwerkverbindungen. Die gewählte Systemarchitektur erlaubt eine breite Anwendbarkeit und lässt sich flexibel an unterschiedliche Nutzungskontexte anpassen.

Das System ermöglicht eine durchgängige Prozesskette, von der Erfassung sensorischer Daten über deren Verarbeitung bis hin zur gezielten Zustellung von Benachrichtigungen an mobile Endgeräte. Damit leistet es einen wesentlichen Beitrag zur Lösung der eingangs beschriebenen Problemstellung.

Für Android-Geräte konnte eine unabhängige Umsetzung mit MQTT realisiert werden. Bei iOS hingegen besteht weiterhin eine Abhängigkeit von APNs, wodurch eine vollständige Unabhängigkeit von Drittanbietern ausgeschlossen ist. In der Praxis zeigte sich MQTT unter stabilen Netzwerkbedingungen zwar performanter als FCM, jedoch offenbarten die Tests auch Schwächen hinsichtlich der Zuverlässigkeit. So kam es vereinzelt zu Situationen, in denen der MQTT-Client nicht mit dem Broker verbunden war. In diesen Fällen konnte er keine Benachrichtigungen empfangen, unabhängig von der Netzqualität. Die angestrebte maximale Reaktionszeit von zehn Sekunden konnte überwiegend eingehalten werden, kann jedoch bei instabiler Netzlage gelegentlich überschritten werden.

8.2 Ausblick auf zukünftige Entwicklungen

Das entwickelte System bietet eine solide Basis für weiterführende Optimierungen und Erweiterungen. Die derzeitige Abhängigkeit von FCM sowie das Fehlen eines plattformübergreifenden SDKs beeinträchtigen den praktischen Nutzen nur geringfügig, weisen jedoch auf weiteres Entwicklungspotenzial hin. Insbesondere das Push-Backend wurde als möglicher Engpass bei steigender Nutzeranzahl identifiziert, was Optimierungen hinsichtlich der Skalierbarkeit nahelegt.

Darüber hinaus bestehen strukturelle Herausforderungen, die eine vollständig plattformunabhängige Lösung ohne proprietäre Dienste erschweren. Die zunehmende Abschottung mobiler Betriebssysteme schränkt die Möglichkeiten unabhängiger Implementierungen deutlich ein. Dennoch lässt sich festhalten, dass ein generisches, modular aufgebautes Benachrichtigungssystem ein wichtiger Schritt in Richtung flexibler, sensorgestützter Anwendungen ist. Insbesondere in Szenarien, in denen schnelle, kontextbezogene Reaktionen gefordert sind.

Literaturverzeichnis

42matters (2025a). *Top 20 Push Notifications SDKs Used in Android Apps on Google Play*. URL: https://42matters.com/sdk-analysis/top-push-notifications-sdks (Zugriff am 23. 03. 2025).

– (2025b). *Top 20 Push Notifications SDKs Used in iOS Apps on the Apple App Store*. URL: https://42matters.com/sdk-analysis/app-store/top-push-notifications-sdks (Zugriff am 23. 03. 2025).

Adhyaru, B. B., Hilburn, G., Oberg, M., Mann, K. u. a. (2023). „Push notifications for critical labs results: a pilot study in the intensive care unit (ICU)“. In: *JAMIA Open* 6.3, ooad058. ISSN: 2574-2531. DOI: 10.1093/jamiaopen/ooad058. eprint: https://academic.oup.com/jamiaopen/article-pdf/6/3/ooad058/51119522/ooad058.pdf. URL: https://doi.org/10.1093/jamiaopen/ooad058.

Adler, J. M. (2022). *Firebase Cloud Messaging (FCM) Compared to OneSignal*. URL: https://onesignal.com/blog/firebase-vs-onesignal/ (Zugriff am 10. 02. 2025).

Agrawal, S. (2022). *How many requests per min we can make to send messages api call for a notification hub ?* URL: https://learn.microsoft.com/en-us/answers/questions/783989/how-many-requests-per-min-we-can-make-to-send-mess (Zugriff am 01. 05. 2025).

Android Developer (2025). *Foreground services overview*. URL: https://developer.android.com/develop/background-work/services/fgs (Zugriff am 21. 04. 2025).

Apple Developer (2025). *Setting up a remote notification server*. URL: https://developer.apple.com/documentation/usernotifications/setting-up-a-remote-notification-server (Zugriff am 25. 04. 2025).

Apple Engineer (2021). *Apple Developer Forums - Keep ios app running in background forever*. URL: https://developer.apple.com/forums/thread/681789?answerId=677801022#677801022 (Zugriff am 04. 04. 2025).

aws (2025). *Was ist ein API-Schlüssel?* URL: https://aws.amazon.com/de/what-is/api-key/ (Zugriff am 23. 04. 2025).

Balzert, H. (2009). *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. ger. 3. Auflage. SpringerLink Bücher. Heidelberg: Spektrum Akademischer Verlag, S. 456. ISBN: 9783827422477. DOI: 10.1007/978-3-8274-2247-7. URL: https://doi.org/10.1007/978-3-8274-2247-7.

Born, W. & Weitkamp, C. (2019). *License - openHAB Distro*. URL: https://github.com/openhab/openhab-distro/blob/main/LICENSE (Zugriff am 22. 01. 2025).

Burgstahler, D., Richerzhagen, N., Englert, F., Hans, R. u. a. (2014). „Switching Push and Pull: An Energy Efficient Notification Approach“. In: *2014 IEEE International Conference on Mobile Services*, S. 68–75. DOI: 10.1109/MobServ.2014.19.

Cloudflare (2025). *What is the cloud? | Cloud definition*. URL: https://www.cloudflare.com/learning/cloud/what-is-the-cloud/ (Zugriff am 01. 05. 2025).

Deglin, G. (2018). *Product & Policy Updates for GDPR*. URL: https://onesignal.com/blog/product-policy-updates-for-gdpr/ (Zugriff am 06. 03. 2025).

Eka Putra, F. P., Muslim, F., Hasanah, N., Holipah u. a. (2024). „Analisis Komparasi Protokol Websocket dan MQTT Dalam Proses Push Notification“. In: *Jurnal Sistim Informasi dan Teknologi* 5.4, S. 63–72. DOI: 10.60083/jsisfotek.v5i4.325. URL: https://www.jsisfotek.org/index.php/JSisfotek/article/view/325.

Firebase (2023). *FirebaseApp*. URL: https://firebase.google.com/docs/reference/kotlin/com/google/firebase/FirebaseApp (Zugriff am 26. 04. 2025).

– (2024a). *Datenschutz und Sicherheit in Firebase*. URL: https://firebase.google.com/support/privacy?hl=de#firebase_is_certified_under_major_privacy_and_security_standards (Zugriff am 10. 02. 2025).

– (2024b). *Firebase Cloud Messaging*. URL: https://firebase.google.com/docs/cloud-messaging (Zugriff am 10. 02. 2025).

– (2025a). *Best Practices beim Senden von FCM-Nachrichten in großem Umfang*. URL: https://firebase.google.com/docs/cloud-messaging/scale-fcm?hl=de (Zugriff am 10. 02. 2025).

– (2025b). *FCM Architectural Overview*. URL: https://firebase.google.com/docs/cloud-messaging/fcm-architecture (Zugriff am 08. 04. 2025).

– (2025c). *Firebase Status Dashboard*. URL: https://status.firebase.google.com/products/K2T198LhzrR5kLU8hmkk/history (Zugriff am 10. 02. 2025).

– (2025d). *Ihre Serverumgebung und FCM*. URL: https://firebase.google.com/docs/cloud-messaging/server (Zugriff am 27. 04. 2025).

– (2025e). *Pricing plans*. URL: https://firebase.google.com/pricing (Zugriff am 10. 02. 2025).

– (2025f). *Release Notes*. URL: https://firebase.google.com/support/releases (Zugriff am 10. 04. 2025).

– (2025g). *SDKs und Clientbibliotheken*. URL: https://firebase.google.com/docs/firestore/client/libraries?hl=de (Zugriff am 10. 02. 2025).

Fischer, I. (2024). *ioBroker pushsafer Adapter*. URL: https://github.com/ioBroker/ioBroker.pushsafer (Zugriff am 14. 04. 2025).

– (2025). *ioBroker GitHub Releases*. URL: https://github.com/ioBroker/ioBroker/releases (Zugriff am 10. 04. 2025).

Ganiger, R., Kanjarbhat, P., Hunagund, S., Ramesh, A. K. u. a. (2024). „IoT based protection and monitoring system for induction motor“. In: *AIP Conference Proceedings* 3131.1, S. 030002. ISSN: 0094-243X. DOI: 10.1063/5.0229698. eprint: https://pubs.aip.org/aip/acp/article-pdf/doi/10.1063/5.0229698/20165649/030002_1_5.0229698.pdf. URL: https://doi.org/10.1063/5.0229698.

GitHub (2025). *About GitHub and Git*. URL: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git (Zugriff am 26. 04. 2025).
GitHub Staff (2024). *The State of Open Source - Octoverse 2024*. URL: https://github.blog/news-insights/octoverse/octoverse-2024/#the-state-of-open-source (Zugriff am 22. 01. 2025).
Google (2025). *Leistungsstarke Codierung, Block für Block*. URL: https://developers.google.com/blockly?hl=de (Zugriff am 13. 03. 2025).
Google Play (2025a). *Google Play - Gotify*. URL: https://play.google.com/store/apps/details?id=com.github.gotify&hl=de (Zugriff am 04. 02. 2025).
– (2025b). *Google Play - Pushbullet*. URL: https://play.google.com/store/apps/details?id=com.pushbullet.android&hl=de (Zugriff am 04. 02. 2025).
– (2025c). *Google Play - Pushover*. URL: https://play.google.com/store/apps/details?id=net.superblock.pushover&hl=de (Zugriff am 04. 02. 2025).
– (2025d). *Google Play - Pushsafer*. URL: https://play.google.com/store/apps/details?id=de.appzer.Pushsafer&hl=de (Zugriff am 04. 02. 2025).
Google Trends (2025). *Interesse im zeitlichen Verlauf*. URL: https://trends.google.de/trends/explore?date=2012-01-01%202025-01-23&q=%2Fg%2F11gk8wrw1j,%2Fg%2F1yw9kr31z,%2Fg%2F11fzxlb_q4&hl=de (Zugriff am 23. 01. 2025).
Gotify (2025a). *Gotify*. URL: https://gotify.net (Zugriff am 06. 03. 2025).
– (2025b). *Gotify API Documentation*. Version 2.0.2. URL: https://gotify.net/api-docs (Zugriff am 16. 01. 2025).
– (2025c). *gotify/server*. URL: https://github.com/gotify/server (Zugriff am 04. 02. 2025).
– (2025d). *gotify/server*. URL: https://gotify.net/img/ui.png (Zugriff am 04. 03. 2025).
Haev, D. (2024a). *ioBroker.pushover*. URL: https://github.com/ioBroker/ioBroker.pushover (Zugriff am 14. 04. 2025).
– (2024b). *MIT License - ioBroker*. URL: https://github.com/ioBroker/ioBroker.build/blob/master/LICENSE (Zugriff am 21. 01. 2025).
Halvorsen, H.-P., Grytten, O. A., Svendsen, M. V. & Mylvaganam, S. (2018). „Environmental Monitoring with Focus on Emissions Using IoT Platform for Mobile Alert“. In: *2018 28th EAEEIE Annual Conference (EAEEIE)*, S. 1–7. DOI: 10.1109/EAEEIE.2018.8534197.
Haxhibeqiri, J., De Poorter, E., Moerman, I. & Hoebeke, J. (2018). „A Survey of LoRaWAN for IoT: From Technology to Application“. In: *Sensors* 18.11. ISSN: 1424-8220. DOI: 10.3390/s18113995. URL: https://www.mdpi.com/1424-8220/18/11/3995.
Home Assistant (2018). *The Apache 2.0 License - Home Assistant Developers*. Version 2015.1.3. URL: https://www.home-assistant.io/developers/license/ (Zugriff am 22. 01. 2025).
– (2022). *Frontend of Home Assistant*. Version 2015.1.3. URL: https://www.home-assistant.io/docs/frontend/ (Zugriff am 22. 01. 2025).
– (2025a). *Adding integrations*. URL: https://www.home-assistant.io/getting-started/integration/ (Zugriff am 21. 01. 2025).

Home Assistant (2025b). *Automating Home Assistant*. URL: https://www.home-assistant.io/docs/automation/ (Zugriff am 21. 01. 2025).
– (2025c). *Creating your first integration*. URL: https://developers.home-assistant.io/docs/creating_component_index/ (Zugriff am 14. 04. 2025).
– (2025d). *FAQ - Release Schedule - Home Assistant*. Version 2015.1.3. URL: https://www.home-assistant.io/faq/release/ (Zugriff am 22. 01. 2025).
– (2025e). *Home Assistant - Awaken your home*. Version 2015.1.3. URL: https://www.home-assistant.io/ (Zugriff am 22. 01. 2025).
ioBroker (2024a). *ioBroker - Automate your life*. URL: https://www.iobroker.net/ (Zugriff am 17. 04. 2025).
– (2024b). *ioBroker Dokumentation*. URL: https://www.iobroker.net/#de/documentation (Zugriff am 21. 01. 2025).
– (2025a). *ioBroker Adapter's List*. URL: https://download.iobroker.net/list.html (Zugriff am 07. 01. 2025).
– (2025b). *Statistics*. URL: https://www.iobroker.net/#de/statistics (Zugriff am 22. 01. 2025).
IONOS (2023). *Was ist ein API-Key?* URL: https://www.ionos.de/digitalguide/websites/web-entwicklung/api-key/ (Zugriff am 23. 04. 2025).
Kärkkäinen, J. (2025). *Was ist eine Integrationsplattform? Typen, Beispiele und Services, die Sie von ITSM in Betracht ziehen sollten*. URL: https://www.oneio.cloud/de/blog/what-is-an-integration-platform (Zugriff am 09. 04. 2025).
Kleuker, S. (2013). „Anforderungsanalyse“. In: *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten*. Wiesbaden: Springer Fachmedien Wiesbaden, S. 55. ISBN: 978-3-658-00642-6. DOI: 10.1007/978-3-658-00642-6_4. URL: https://doi.org/10.1007/978-3-658-00642-6_4.
Langer, C. (2015). *Pushbullet: Ende-zu-Ende-Verschlüsselung*. URL: https://linuxundich.de/android/pushbullet-ende-zu-ende-verschluesselung/ (Zugriff am 16. 01. 2025).
Langholz, S. (2021). *Guide to Understanding Push Notification Performance*. URL: https://onesignal.com/blog/guide-to-understanding-push-notification-performance/ (Zugriff am 10. 02. 2025).
Łasocha, W. & Badurowicz, M. (2021). „Comparison of WebSocket and HTTP protocol performance“. In: *Journal of Computer Sciences Institute* 19, S. 67–74. DOI: 10.35784/jcsi.2452. URL: https://ph.pollub.pl/index.php/jcsi/article/view/2452.
LoRaWAN (2025). *Was ist LoRaWAN?* URL: https://www.lora-wan.de/ (Zugriff am 29. 04. 2025).
Lv, R., Feng, D., Jiao, X., Wu, G. u. a. (2024). „Research and Development of Real-time Pushing System based on HTTP Long Polling“. In: *2024 7th International Conference on Computer Information Science and Application Technology (CISAT)*, S. 644–647. DOI: 10.1109/CISAT62382.2024.10695284.
M., M. (2024). *ioBroker pushbullet Adapter*. URL: https://github.com/iobroker-community-adapters/ioBroker.pushbullet (Zugriff am 14. 04. 2025).
Mattheis, J. (2025). *server/License*. URL: https://github.com/gotify/server/blob/master/LICENSE (Zugriff am 04. 02. 2025).

Microsoft (2014). *https Protocol*. URL: https://learn.microsoft.com/en-us/previous-versions/aa767735(v=vs.85) (Zugriff am 29. 04. 2025).
– (2020). *Azure Notification Hubs SDKs*. URL: https://learn.microsoft.com/en-us/azure/notification-hubs/notification-hubs-sdks (Zugriff am 10. 02. 2025).
– (2021). *Tutorial: Set up your iOS app to work with Azure Notification Hubs*. URL: https://learn.microsoft.com/en-us/azure/notification-hubs/ios-sdk-get-started (Zugriff am 08. 04. 2025).
– (2023). *Sicherheit von Notification Hubs*. URL: https://learn.microsoft.com/de-de/azure/notification-hubs/notification-hubs-push-notification-security (Zugriff am 10. 02. 2025).
– (2024a). *Reliability in Azure Notification Hubs - Availability zone support*. URL: https://learn.microsoft.com/en-us/azure/reliability/reliability-notification-hubs#availability-zone-support (Zugriff am 10. 02. 2025).
– (2024b). *Tutorial: Send push notifications to Android devices using Firebase SDK version 1.0.0-preview1*. URL: https://learn.microsoft.com/de-de/azure/notification-hubs/android-sdk (Zugriff am 08. 04. 2025).
– (2025a). *App Service-Tarifplan auswählen*. URL: https://portal.azure.com/?quickstart=True#view/WebsitesExtension/ScaleSpecPicker.ReactView/id/%2Fsubscriptions%2Fbc98bf5f-a73f-475a-bae2-7e1bac386fb8%2FresourceGroups%2FPushNotifier_Test%2Fproviders%2FMicrosoft.Web%2FserverFarms%2Fasp-push-PushNotifierTest (Zugriff am 17. 02. 2025).
– (2025b). *Choose your pricing tier*. URL: https://portal.azure.com/?quickstart=True#create/Microsoft.NotificationHub (Zugriff am 24. 01. 2025).
– (2025c). *Notification Hubs*. URL: https://azure.microsoft.com/de-de/products/notification-hubs (Zugriff am 10. 02. 2025).
– (2025d). *Notification Hubs pricing*. URL: https://azure.microsoft.com/en-us/pricing/details/notification-hubs/ (Zugriff am 10. 02. 2025).
– (2025e). *Push notifications with Azure Notification Hubs: Frequently asked questions*. URL: https://learn.microsoft.com/en-us/azure/notification-hubs/notification-hubs-push-notification-faq#is-there-any-latency-guarantee- (Zugriff am 10. 02. 2025).
– (2025f). *Push notifications with Azure Notification Hubs: Frequently asked questions*. URL: https://learn.microsoft.com/en-us/azure/notification-hubs/notification-hubs-push-notification-faq#where-does-azure-notification-hubs-store-data- (Zugriff am 10. 02. 2025).
– (2025g). *SQL Datenbank Konfigurieren*. URL: https://portal.azure.com/?quickstart=True#create/Microsoft.SQLDatabase (Zugriff am 18. 02. 2025).
Moroney, L. (2017). „Firebase Cloud Messaging“. In: *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform*. Berkeley, CA: Apress, S. 163–188. ISBN: 978-1-4842-2943-9. DOI: 10.1007/978-1-4842-2943-9_9. URL: https://doi.org/10.1007/978-1-4842-2943-9_9.
MQTT (2024). *MQTT: The Standard for IoT Messaging*. URL: https://mqtt.org/ (Zugriff am 03. 04. 2025).
NABU CASA (2025a). *Pricing*. URL: https://www.nabucasa.com/pricing/ (Zugriff am 10. 04. 2025).

NABU CASA (2025b). *Privacy*. URL: https://www.nabucasa.com/privacy/ (Zugriff am 10. 04. 2025).
OneSignal (2023). *GDPR & Individual Rights*. URL: https://documentation.onesignal.com/docs/gdpr-compliance (Zugriff am 10. 02. 2025).
– (2024a). *Android: Firebase Credentials*. URL: https://documentation.onesignal.com/docs/android-firebase-credentials (Zugriff am 10. 02. 2025).
– (2024b). *Huawei SDK setup*. URL: https://documentation.onesignal.com/docs/huawei-sdk-setup (Zugriff am 08. 04. 2025).
– (2024c). *Mobile SDK Setup*. URL: https://documentation.onesignal.com/docs/mobile-sdk-setup (Zugriff am 10. 02. 2025).
– (2024d). *Web SDK setup*. URL: https://documentation.onesignal.com/docs/web-sdk-setup (Zugriff am 08. 04. 2025).
– (2025a). *About OneSignal*. URL: https://onesignal.com/about (Zugriff am 17. 02. 2025).
– (2025b). *iOS SDK setup*. URL: https://documentation.onesignal.com/docs/ios-sdk-setup (Zugriff am 08. 04. 2025).
– (2025c). *Maximize engagement, minimize churn*. URL: https://onesignal.com (Zugriff am 17. 02. 2025).
– (2025d). *Quickstart guide*. URL: https://documentation.onesignal.com/docs/quickstart-guide (Zugriff am 01. 05. 2025).
– (2025e). *Rate limits*. URL: https://documentation.onesignal.com/reference/rate-limits (Zugriff am 01. 05. 2025).
– (2025f). *System status*. URL: https://status.onesignal.com/ (Zugriff am 11. 02. 2025).
openHAB (2025a). *Add-ons - openHAB*. Version 4.3.2. URL: https://www.openhab.org/addons/ (Zugriff am 22. 01. 2025).
– (2025b). *Developer Documentation - openHAB*. Version 4.3.2. URL: https://www.openhab.org/docs/developer/ (Zugriff am 22. 01. 2025).
– (2025c). *Developing a Binding*. URL: https://www.openhab.org/docs/developer/bindings/ (Zugriff am 21. 01. 2025).
– (2025d). *GitHub - openHAB Core Tags*. URL: https://github.com/openhab/openhab-core/tags (Zugriff am 10. 04. 2025).
– (2025e). *Installation of Add-ons*. URL: https://www.openhab.org/docs/configuration/addons.html (Zugriff am 14. 04. 2025).
– (2025f). *openHAB - empowering the smart home*. URL: https://www.openhab.org (Zugriff am 10. 04. 2025).
– (2025g). *openHAB Community Forum*. URL: https://community.openhab.org/ (Zugriff am 10. 04. 2025).
– (2025h). *Rules*. URL: https://www.openhab.org/docs/concepts/rules.html (Zugriff am 21. 01. 2025).
– (2025i). *User Interface Design Overview*. Version 4.3.2. URL: https://www.openhab.org/docs/ui/ (Zugriff am 21. 01. 2025).
Özdil, E. (2025). *API*. URL: https://www.weclapp.com/de/lexikon/api/ (Zugriff am 22. 04. 2025).

Pohl, T. (2024). *ioBroker.gotify – Adapter zur Integration von Gotify-Benachrichtigungen*. URL: https://github.com/ThomasPohl/ioBroker.gotify (Zugriff am 14. 04. 2025).
Postman (2024). *Test your API using the Collection Runner*. URL: https://learning.postman.com/docs/collections/running-collections/intro-to-collection-runs/ (Zugriff am 23. 04. 2025).
– (2025). *Your Complete API Platform, From Design to Delivery*. URL: https://www.postman.com/ (Zugriff am 23. 04. 2025).
Pushbullet (2020a). *Push Limit*. URL: https://docs.pushbullet.com/#push-limit (Zugriff am 16. 01. 2025).
– (2020b). *Pushbullet*. URL: https://www.pushbullet.com/ (Zugriff am 06. 03. 2025).
– (2020c). *Pushbullet API Documentation*. URL: https://docs.pushbullet.com/#pushbullet-api (Zugriff am 16. 01. 2025).
– (2025a). *Pushbullet - Integration Home Assistant*. URL: https://www.home-assistant.io/integrations/pushbullet/ (Zugriff am 14. 04. 2025).
– (2025b). *Pushbullet Binding*. URL: https://www.openhab.org/addons/bindings/pushbullet/ (Zugriff am 14. 04. 2025).
– (2025c). *Welcome to Pushbullet!* URL: https://www.pushbullet.com/#setup (Zugriff am 04. 03. 2025).
PushNotifier (2023a). *PushNotifier Api - Get Devices*. URL: https://api.pushnotifier.de/v2/doc/#endpoint-get-get-devices (Zugriff am 13. 02. 2025).
– (2023b). *PushNotifier API Documentation v2*. URL: https://api.pushnotifier.de/v2/doc/ (Zugriff am 13. 02. 2025).
– (2025a). *Add Device*. URL: https://pushnotifier.de/devices/add (Zugriff am 13. 02. 2025).
– (2025b). *Datenschutzerklärung*. URL: https://pushnotifier.de/privacy (Zugriff am 13. 02. 2025).
– (2025c). *Help*. URL: https://pushnotifier.de/help (Zugriff am 27. 03. 2025).
– (2025d). *PushNotifier*. URL: https://pushnotifier.de/ (Zugriff am 13. 02. 2025).
– (2025e). *PushNotifier Devices*. URL: https://pushnotifier.de/devices (Zugriff am 13. 02. 2025).
– (2025f). *PushNotifier Login*. URL: https://pushnotifier.de/login (Zugriff am 13. 02. 2025).
– (2025g). *Send Notification*. URL: https://pushnotifier.de/notifications (Zugriff am 13. 02. 2025).
Pushover (2024). *Pushover - Simple Notifications*. URL: https://pushover.net/ (Zugriff am 04. 02. 2025).
– (2025a). *Pushover - Integration Home Assistant*. URL: https://www.home-assistant.io/integrations/pushover/ (Zugriff am 14. 04. 2025).
– (2025b). *Pushover API Documentation*. URL: https://pushover.net/api (Zugriff am 16. 01. 2025).
– (2025c). *Pushover Binding*. URL: https://www.openhab.org/addons/bindings/pushover/ (Zugriff am 14. 04. 2025).
– (2025d). *Pushover for Desktop*. URL: https://pushover.net/clients/desktop (Zugriff am 19. 03. 2025).

Pushover (2025e). *Pushover Open Client API*. URL: https://pushover.net/api/client (Zugriff am 03. 03. 2025).
– (2025f). *Pushover Status*. URL: https://status.pushover.net/ (Zugriff am 07. 03. 2025).
– (2025g). *Pushover-Powered Applications & Plugins*. URL: https://pushover.net/apps (Zugriff am 04. 02. 2025).
Pushover-Support (2021). *Purchasing Additional Capacity to Send More Messages per Month*. URL: https://support.pushover.net/i13-purchasing-additional-capacity-to-send-more-messages-per-month (Zugriff am 16. 01. 2025).
Pushsafer (2025a). *Beispiele & Plugins*. URL: https://www.pushsafer.com/de/examples (Zugriff am 04. 02. 2025).
– (2025b). *FAQ*. URL: https://www.pushsafer.com/de/faq#answer7 (Zugriff am 16. 01. 2025).
– (2025c). *FAQ*. URL: https://www.pushsafer.com/de/faq#answer2 (Zugriff am 16. 01. 2025).
– (2025d). *Nachrichten senden*. URL: https://www.pushsafer.com/de/index (Zugriff am 03. 05. 2025).
– (2025e). *Pushsafer*. URL: https://www.pushsafer.com/de/index (Zugriff am 04. 02. 2025).
– (2025f). *Pushsafer*. URL: https://www.pushsafer.com/dashboard (Zugriff am 04. 03. 2025).
– (2025g). *Pushsafer - Integration Home Assistant*. URL: https://www.home-assistant.io/integrations/pushsafer/ (Zugriff am 14. 04. 2025).
– (2025h). *Pushsafer Binding*. URL: https://www.openhab.org/addons/bindings/pushsafer/ (Zugriff am 14. 04. 2025).
– (2025i). *Pushsafer Forum*. URL: https://www.pushsafer.com/board/viewforum.php?f=24 (Zugriff am 04. 02. 2025).
Pushy (2023). *What is MQTT?* URL: https://support.pushy.me/hc/en-us/articles/360043864271-What-is-MQTT (Zugriff am 15. 04. 2025).
– (2024). *pushy / pushy-demo-android Main.java*. URL: https://github.com/pushy/pushy-demo-android/blob/master/app/src/main/java/me/pushy/example/Main.java (Zugriff am 16. 04. 2025).
Rosencrance, L. (2017). *Zigbee*. URL: https://www.techtarget.com/iotagenda/definition/ZigBee (Zugriff am 29. 04. 2025).
Saeed, F., Paul, A., Rehman, A., Hong, W. H. u. a. (2018). „IoT-Based Intelligent Modeling of Smart Home Environment for Fire Prevention and Safety“. In: *Journal of Sensor and Actuator Networks* 7.1. ISSN: 2224-2708. DOI: 10.3390/jsan7010011. URL: https://www.mdpi.com/2224-2708/7/1/11.
Sawant, P. (2020). *Apple Developer - Build local push connectivity for restricted networks*. URL: https://developer.apple.com/videos/play/wwdc2020/10113/ (Zugriff am 04. 04. 2025).
Sebastian (2025). *GitHub - iGotify Notification Assistent*. URL: https://github.com/androidseb25/iGotify-Notification-Assistent (Zugriff am 09. 04. 2025).
simatec (2025). *ioBroker.gotify-ws*. URL: https://github.com/simatec/ioBroker.gotify-ws (Zugriff am 14. 04. 2025).

Stein, J. (2024). *News and updates for Pushover Notifications*. URL: https://blog.pushover.net/ (Zugriff am 09. 04. 2025).

Subahi, A. F. (2023). „BERT-Based Approach for Greening Software Requirements Engineering Through Non-Functional Requirements“. In: *IEEE Access* 11, S. 103001–103013. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3317798.

Tang, K., Wang, Y., Liu, H., Sheng, Y. u. a. (2013). „Design and Implementation of Push Notification System Based on the MQTT Protocol“. In: *Proceedings of 2013 International Conference on Information Science and Computer Applications*. Atlantis Press, S. 116–119. ISBN: 978-90786-77-85-7. DOI: 10.2991/isca-13.2013.20. URL: https://doi.org/10.2991/isca-13.2013.20.

Timonera, K. (2023). *What Is API Security? Definition, Fundamentals, & Tips*. URL: https://www.esecurityplanet.com/applications/api-security/ (Zugriff am 29. 04. 2025).

WERTGARANTIE (2025). *OpenHAB - Smart Home Lexikon*. URL: https://www.wertgarantie.de/lexikon/smart-home/openhab (Zugriff am 22. 01. 2025).

Yasar, K. (2022). *data transfer rate (DTR)*. URL: https://www.techtarget.com/searchunifiedcommunications/definition/data-transfer-rate (Zugriff am 26. 04. 2025).

Yasar, K. & Gillis, A. S. (2024). *internet of things (IoT)*. URL: https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT (Zugriff am 29. 04. 2025).

Yasar, K. & Rosencrance, L. (2022). *software development kit (SDK)*. URL: https://www.techtarget.com/whatis/definition/software-developers-kit-SDK (Zugriff am 26. 04. 2025).

Z-WAVE Alliance (2025). *Technology Overview*. URL: https://z-wavealliance.org/technology-overview/ (Zugriff am 29. 04. 2025).

Zhang, L. & Shen, X. (2013). „Research and development of real-time monitoring system based on WebSocket technology“. In: *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, S. 1955–1958. DOI: 10.1109/MEC.2013.6885373.

Zheng, Y. & Wang, Q. R. (2020). „Shadow of the Great Firewall: The Impact of Google Blockade on Innovation in China“. In: *Strategic Management Journal* 41.12, S. 2260–2294. DOI: 10.1002/smj.3179. URL: https://doi.org/10.1002/smj.3179.

A Anhang

A.1 Tabellen

Tabelle A.1: Relevante Anforderungskriterien zur systematischen Gegenüberstellung der Systemkomponenten.

Anforderung	**Integrations-plattform**	**Bestehendes System**	**Push-Dienst**	**Push-Backend**
Verarbeitung der Sensordaten (FA 1)	✓		(✓)	(✓)
Speicherung der Benachrichtigungen (FA 2)		✓		✓
Mandantenfähigkeit (FA 3)		✓		✓
Geräteverwaltung (FA 4)		✓		✓
Integration in Anwendungen (FA 5)		(✓)	✓	(✓)
Zuverlässigkeit (NFA 1)	k.A.	k.A.	✓	✓
Skalierbarkeit (NFA 2)	✓	✓	✓	✓
Reaktionszeiten (NFA 3)	✓	✓	✓	✓
Wartbarkeit und Erweiterbarkeit (NFA 4)	✓	✓	✓	✓
Ressourcennutzung (NFA 5)	✓	✓	Cloud	✓

Anforderung	Integrations-plattform	Bestehendes System	Push-Dienst	Push-Backend
Aktualität und Zukunftsfähigkeit (NFA 6)	✓	✓	✓	✓
Externe Dienste (NFA 7)	✓	✓	✓	✓
Datenschutz (NFA 8)	✓	✓	✓	✓
Benutzerfreundlichkeit (NFA 9)	✓	✓		✓
Kompatibilität (NFA 10)	✓	✓		
Portabilität (NFA 11)		✓	✓	✓

A.2 Quellcode ioBroker Adapter

Listing A.1: Hauptdatei des Adapters

```
1  const utils = require('@iobroker/adapter-core');
2  const axios = require('axios');
3
4  const API_URL = 'https://[...]/api/';
5  const SEND_TO_USER = 'Push/SendToUser';
6
7  class SensorAdapter extends utils.Adapter {
8      constructor(options = {}) {
9          super({ ...options, name: 'pushnotifier' });
10
11         this.on('ready', this.onReady.bind(this));
12         this.on('unload', this.onUnload.bind(this));
13         this.on('message', this.onMessage.bind(this));
14     }
15
16     async onReady() {
17         this.log.debug(`config token: ${this.config.token}`);
18
19         if (this.config.token) {
20             await this.setState('info.connection', true, true
                   );
21             this.log.info('Adapter␣configured');
22         } else {
```

```
23              await this.setState('info.connection', false,
                    true);
24              this.log.warn('Adapter not configured');
25          }
26      }
27
28      async sendPushNotification(notificationMessage) {
29          const apiUrl = API_URL + SEND_TO_USER;
30          const token = this.config.token;
31
32          if (!token) {
33              this.log.error('API token not configured in the
                    adapter settings.');
34              return;
35          }
36
37          const data = {
38              title: notificationMessage.title || this.config.
                    title || "Warning",
39              text: notificationMessage.message || this.config.
                    message || "Sensor was triggered",
40              type: "Web",
41              imageUrl : this.config.imageUrl || null,
42              apiKey: token,
43              payload: {
44                  appDeepLink: "sensorAlert",
45                  event: "sensorTriggered"
46              }
47          };
48
49          try {
50              const response = await axios.post(apiUrl, data);
51              this.log.info(`API notification sent successfully
                    . Response: ${response.status}`);
52          } catch (error) {
53              this.log.error(`Error sending the API
                    notification: ${error.message}`);
54          }
55      }
56
57      onMessage(obj) {
58          if (!obj || typeof obj !== 'object' || !obj.command)
                {
59              this.log.warn('Invalid message received!');
60              return;
61          }
62
63          if (obj.command === 'send') {
64              this.sendPushNotification(obj.message);
```

```
65          } else {
66              this.log.warn(`Unknown command: ${obj.command}`);
67          }
68      }
69
70      onUnload(callback) {
71          try {
72              this.log.info('Adapter stopped.');
73              callback();
74          } catch (e) {
75              this.log.error(`Error when stopping: ${e}`);
76              callback();
77          }
78      }
79  }
80
81  if (require.main !== module) {
82      module.exports = (options) => new SensorAdapter(options);
83  } else {
84      new SensorAdapter();
85  }
```

Listing A.2: io-package.json

```
1  {
2          "common": {
3                  "name": "pushnotifier",
4                  "version": "0.0.1",
5                  "titleLang": {
6                          "en": "PushNotifier"
7                  },
8                  "desc": {
9                          "en": "This adapter allows to send
                             pushnotifier notifications from
                             ioBroker"
10                 },
11                 "platform": "Javascript/Node.js",
12                 "icon": "pushnotifier.png",
13                 "enabled": true,
14                 "loglevel": "info",
15                 "tier": 2,
16                 "mode": "daemon",
17                 "type": "messaging",
18                 "compact": true,
19                 "connectionType": "cloud",
20                 "dataSource": "push",
21                 "adminUI": {
22                         "config": "materialize"
23                 },
24                 "blockly": true,
```

```
                "messagebox": true,
                "dependencies": [
                        {
                                "js-controller": ">=6.0.11"
                        }
                ],
                "globalDependencies": [
                        {
                                "admin": ">=7.0.23"
                        }
                ],
                "supportedMessages": {
            "custom": true,
            "notifications": true
        }
        },
        "native": {
                "token": "",
                "title": "Warning",
                "message": "Sensor was triggered",
                "imageUrl": ""
        },
        "objects": [],
        "instanceObjects": [
                {
                        "_id": "info",
                        "type": "channel",
                        "common": {
                                "name": "Information"
                        },
                        "native": {}
                },
                {
                        "_id": "info.connection",
                        "type": "state",
                        "common": {
                                "role": "indicator.connected"
                                   ,
                                "name": "Device or service
                                   connected",
                                "type": "boolean",
                                "read": true,
                                "write": false,
                                "def": false
                        },
                        "native": {}
                }
        ]
}
```

Listing A.3: Beispiel eines Ausführungsscriptes in TypeScript

```
declare namespace ioBroker {
    interface Object {
      state: state;
    }

    interface state {
      val: any;
    }
  }

  on('0_userdata.0.Feuermelder', 'ne', (obj: ioBroker.Object)
      => {
      if (obj.state.val === true) {
          sendTo("pushnotifier.1", "send", {
              title: "Feuermeldung!",
              message: "Der␣Feuermelder␣wurde␣ausgeloest."
          });
      }
  });
```

Abbildung A.1: Beispiel eines Ausführungsscriptes in Rules.

Listing A.4: Testskript zur Untersuchung des Durchsatzes

```
on({id: '0_userdata.0.Feuermelder', change: 'ne'}, function (
    obj) {
    if (obj.state.val === true) {
        console.log("START")
        for(var i = 1; i <= 100; i++){
            sendTo("easypush.1", "send", {
                title: `Feuermeldung ${i}!`,
                message: "Der Feuermelder wurde ausgeloest."
            });
        }
        console.log("ENDE")
    }
});
```

A.3 Weitere Bilder

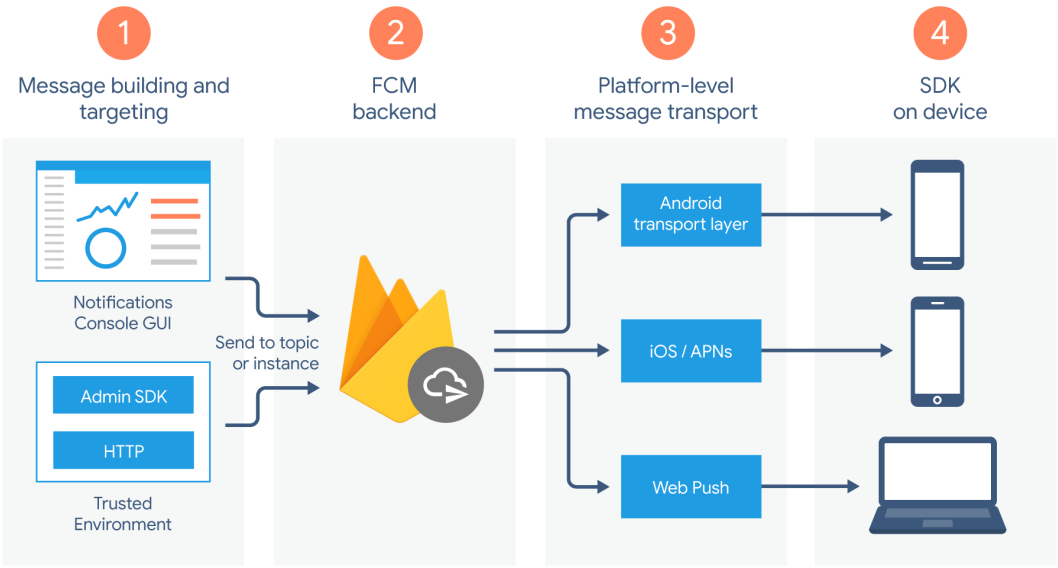

Abbildung A.2: Architektur von FCM.

Abbildung A.3: Benachrichtigungen in der Test-App.

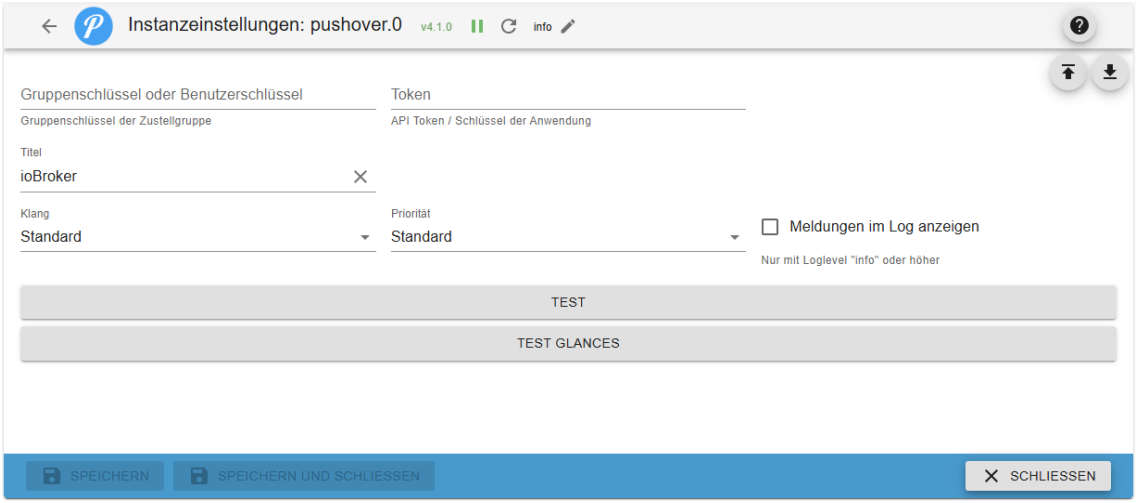

Abbildung A.4: Instanzeinstellungen in ioBroker bei Pushover.

Abbildung A.5: Benachrichtigungen in der MQTT Client App.

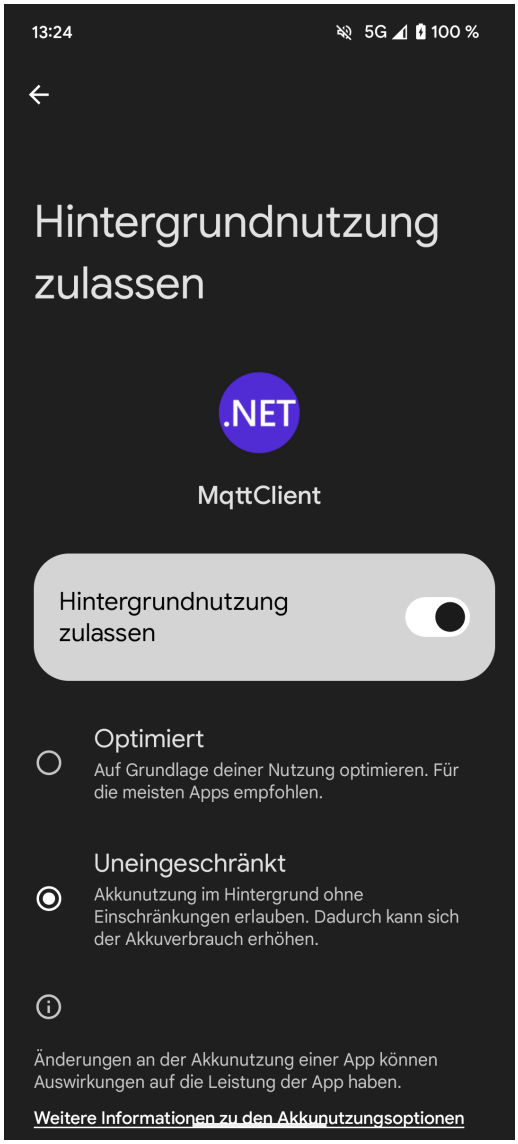

Abbildung A.6: Hintergrundnutzung für die MQTT Client App zuslassen.

A.4 Messdaten

A.4.1 Benachrichtigungssystem mit FCM

Tabelle A.2: Erfasste Latenzzeiten des Systems unter optimalen Netzwerkbedingungen (erhoben am 21. März 2025).

Netzwerkverbindung	1. Sensor gesendet	<-> in s	2. Backend gesendet	<-> in s	3. App Ankunft	Gesamtzeit in s
WLAN	10:17:20,995	0,315	10:17:21,310	0,426	10:17:21,736	0,741
Download 129 Mbit/s	11:48:06,922	0,430	11:48:07,352	1,107	11:48:08,459	1,537
Upload 44 Mbit/s	11:55:04,473	0,318	11:55:04,791	0,898	11:55:05,689	1,216
	12:00:01,780	0,363	12:00:02,143	0,558	12:00:02,701	0,921
	12:01:02,313	0,319	12:01:02,632	0,542	12:01:03,174	0,861
	13:15:42,670	0,458	13:15:43,128	1,131	13:15:44,259	1,589
	14:09:14,609	0,483	14:09:15,092	1,148	14:09:16,240	1,631
	14:14:12,445	0,390	14:14:12,835	0,618	14:14:13,453	1,008
	14:29:54,613	0,349	14:29:54,962	0,625	14:29:55,587	0,974
	15:22:34,569	0,413	15:22:34,982	0,619	15:22:35,601	1,032
Mobiles Netz	15:42:25,962	0,460	15:42:26,422	0,580	15:42:27,002	1,040
Download 60 Mbit/s	15:45:48,930	0,413	15:45:49,343	0,410	15:45:49,753	0,823
Upload 6 Mbit/s	15:50:40,926	0,464	15:50:41,390	0,501	15:50:41,891	0,965
	15:52:27,740	0,434	15:52:28,174	0,584	15:52:28,758	1,018
	16:00:00,478	0,479	16:00:00,957	0,470	16:00:01,427	0,949
	16:33:41,665	0,369	16:33:42,034	0,553	16:33:42,587	0,922
	16:39:45,145	0,346	16:39:45,491	0,458	16:39:45,949	0,804
	16:43:07,015	0,348	16:43:07,363	0,413	16:43:07,776	0,761
	16:44:58,586	0,354	16:44:58,940	0,451	16:44:59,391	0,805
	16:47:31,816	0,350	16:47:32,166	0,407	16:47:32,573	0,757

Hinweis: Die ermittelten Differenzen der Systemuhrzeiten wurden hier schon berücksichtigt.

Tabelle A.3: Erfasste Latenzzeiten des Systems unter Bedingungen eingeschränkter Netzwerkverbindung auf Empfängerseite (erhoben am 01./02. April 2025).

Netzwerkverbindung	1. Sensor gesendet	<-> in s	2. Backend gesendet	<-> in s	3. App Ankunft	Gesamtzeit in s
Mobiles Netz	16:20:08,606	0,459	16:20:09,065	3,231	16:20:12,296	3,690
Download 0,09 Mbit/s	16:27:14,607	0,449	16:27:15,056	3,259	16:27:18,315	3,708
Upload 0,04 Mbit/s	16:31:05,413	0,447	16:31:05,860	1,824	16:31:07,684	2,271
	16:33:18,516	0,597	16:33:19,113	1,231	16:33:20,344	1,828
	16:35:17,260	0,457	16:35:17,717	0,979	16:35:18,696	1,436
	17:00:22,417	0,316	17:00:22,733	0,806	17:00:23,539	1,122
	17:04:02,605	0,311	17:04:02,916	6,903	17:04:09,819	7,214
	17:06:34,213	0,346	17:06:34,559	4,071	17:06:38,630	4,417
	17:28:01,103	0,333	17:28:01,436	2,422	17:28:03,858	2,755
	17:29:55,502	0,333	17:29:55,835	5,799	17:30:01,634	6,132
	17:31:54,731	0,351	17:31:55,082	1,163	17:31:56,245	1,514
	17:34:28,100	0,324	17:34:28,424	0,822	17:34:29,246	1,146
	17:35:59,218	0,351	17:35:59,569	2,131	17:36:01,700	2,482
	17:40:01,786	0,378	17:40:02,164	0,820	17:40:02,984	1,198
	18:11:37,470	0,365	18:11:37,835	0,975	18:11:38,810	1,340
	08:39:42,332	0,425	08:39:42,757	0,671	08:39:43,428	1,096
	08:43:09,409	0,371	08:43:09,780	2,302	08:43:12,082	2,673
	08:46:41,088	0,369	08:46:41,457	2,455	08:46:43,912	2,824
	08:50:02,454	0,366	08:50:02,820	1,621	08:50:04,441	1,987
	08:51:35,498	0,359	08:51:35,857	2,233	08:51:38,090	2,592

Hinweis: Die ermittelten Differenzen der Systemuhrzeiten wurden hier schon berücksichtigt.

Tabelle A.4: Vergleich der Latenzzeiten unter verschiedenen Netzwerkbedingungen (Werte in s).

	Backend Sendet	**App Ankunft**		**Gesamt**	
	gutes Netz	**gutes Netz**	**2G Netz**	**gutes Netz**	**2G Netz**
Anz. Messungen	40	20	20	20	20
min	0,233	0,407	0,671	0,659	1,096
max	0,515	1,148	6,903	1,549	7,214
arithm. Mittel	0,348	0,625	2,286	0,956	2,651
Stdabw. (S)	0,051	0,244	1,691	0,257	1,675

A.4.2 Benachrichtigungssystem mit MQTT

Tabelle A.5: Verbindungs- und Nachrichtenverlauf der Testgeräte mit MQTT-Verbindung

Nr.	**Testgerät**	**Zeitstempel**	**Ereignis**
1	Samsung A14	03.05. 18:00:40	MQTT connected
		03.05. 18:27:26	Feuer los los!
		03.05. 19:47:23	Feuer los los!
		03.05. 21:23:14	Hi
		03.05. 22:16:28	Hi hi
		04.05. 01:54:11	MQTT disconnected
		04.05. 02:00:33	MQTT connected
		04.05. 03:16:36	MQTT disconnected
		04.05. 03:52:31	MQTT connected
		04.05. 05:08:53	MQTT disconnected
		04.05. 05:27:30	MQTT connected
		04.05. 05:50:01	MQTT disconnected
		04.05. 05:56:21	MQTT connected
		04.05. 06:18:08	Guten Morgen
		04.05. 06:18:48	Guten Morgen 2
		04.05. 06:19:12	Guten Morgen 3
		04.05. 07:33:57	MQTT disconnected
		04.05. 08:11:11	MQTT connected
		04.05. 09:03:13	Hallo
		04.05. 09:40:49	MQTT disconnected
		04.05. 09:53:36	MQTT connected
		04.05. 10:36:12	Test 123
		04.05. 11:19:46	MQTT disconnected
		04.05. 11:24:28	MQTT connected
		04.05. 15:39:18	Schokolade
2	Samsung A51	03.05. 17:59:54	MQTT connected
		03.05. 18:27:30	Feuer los los!
		03.05. 19:47:27	Feuer los los!

Nr.	Testgerät	Zeitstempel	Ereignis
2	Samsung A51	03.05. 21:23:18	Hi
		03.05. 22:16:33	Hi hi
		04.05. 06:18:11	Guten Morgen
		04.05. 06:18:52	Guten Morgen 2
		04.05. 06:19:16	Guten Morgen 3
		04.05. 09:03:16	Hallo
		04.05. 10:36:16	Test 123
		04.05. 15:39:23	Schokolade
3	Google Pixel 7a	03.05. 18:02:40	MQTT connected
		03.05. 18:27:31	Feuer los los!
		03.05. 19:47:29	Feuer los los!
		03.05. 21:23:20	Hi
		03.05. 22:16:34	Hi hi
		04.05. 03:22:30	MQTT disconnected
		04.05. 03:30:16	MQTT connected
		04.05. 06:18:12	Guten Morgen
		04.05. 06:18:52	Guten Morgen 2
		04.05. 06:19:16	Guten Morgen 3
		04.05. 07:31:57	MQTT disconnected
		04.05. 07:33:41	MQTT connected
		04.05. 09:03:17	Hallo
		04.05. 10:36:16	Test 123
		04.05. 15:39:23	Schokolade
4	Google Pixel 7a	03.05. 17:46:31	MQTT connected
		03.05. 18:27:31	Feuer los los!
		03.05. 19:47:29	Feuer los los!
		03.05. 21:23:19	Hi
		03.05. 22:16:34	Hi hi
		04.05. 06:18:12	Guten Morgen
		04.05. 06:18:53	Guten Morgen 2
		04.05. 06:19:16	Guten Morgen 3
		04.05. 09:03:17	Hallo
		04.05. 10:36:17	Test 123
		04.05. 15:39:22	Schokolade

Tabelle A.6: Erfasste Latenzzeiten zwischen dem Versand im Backend und der Ankunft in der App über MQTT (erhoben am 18. April 2025).

Netzwerkverbindung	1. Backend gesendet	<-> in s	2. App Ankunft
Mobiles Netz	08:42:58,184	0,196	08:42:58,380
Download 31 Mbit/s	08:44:32,707	0,151	08:44:32,858
Upload 8 Mbit/s	08:48:55,232	0,156	08:48:55,388
	08:51:48,000	0,148	08:51:48,148
	08:53:21,440	0,151	08:53:21,591
	08:56:26,835	0,150	08:56:26,985
	08:57:46,451	0,164	08:57:46,615
	09:00:47,041	0,214	09:00:47,255
	09:01:47,171	0,165	09:01:47,336
	09:03:00,371	0,162	09:03:00,533
	10:40:44,883	0,123	10:40:45,006
	10:41:28,689	0,097	10:41:28,786
	10:42:22,681	0,095	10:42:22,776
	10:43:49,116	0,143	10:43:49,259
	10:44:49,588	0,151	10:44:49,739
	10:44:55,423	0,194	10:44:55,617
	10:45:00,213	0,135	10:45:00,348
	10:45:04,973	0,140	10:45:05,113
	10:45:10,085	0,143	10:45:10,228
	10:45:15,349	0,135	10:45:15,484
Mobiles Netz	09:07:11,654	0,275	09:07:11,929
Download 0,8 Mbit/s	09:08:00,066	0,277	09:08:00,343
Upload 0,4 Mbit/s	09:09:24,452	6,218	09:09:30,670
	09:12:34,707	2,280	09:12:36,987
	09:13:19,305	2,420	09:13:21,725
	09:14:15,060	0,743	09:14:15,803
	09:15:01,687	1,421	09:15:03,108
	09:15:46,340	3,688	09:15:50,028
	09:16:32,451	0,274	09:16:32,725
	09:17:34,239	1,711	09:17:35,950
	10:52:53,053	0,354	10:52:53,407
	10:54:33,946	1,394	10:54:35,340
	10:55:41,056	1,799	10:55:42,855
	10:56:35,789	2,040	10:56:37,829
	10:57:28,606	0,920	10:57:29,526
	10:57:37,166	9,181	10:57:46,347
	10:57:56,729	2,315	10:57:59,044
	10:58:05,022	2,163	10:58:07,185
	10:58:12,548	2,874	10:58:15,422
	10:58:22,214	2,632	10:58:24,846

Hinweis: Die ermittelten Differenzen der Systemuhrzeiten wurden hier schon berücksichtigt.

B Digitaler Anhang auf CD-ROM

B.1 Quellcode

B.1.1 Auszug PushBackend

B.1.2 ioBroker Adapter

B.1.3 MQTT Client App

B.2 PDF-Version der Diplomarbeit

Selbstständigkeitserklärung

Ich versichere, dass ich die Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.