Architekturdokumentation nach Arc42

Semesterprojekt im SS 2023

in Kooperation mit sovanta AG

Produkt “Rocket Deployer”

Team leap

Version 1.0

16.05.2023

Verantwortliche

Yan Wittmann, Lauritz Fuchs

hochschule mannheim

Versionsverzeichnis

Die im Folgenden aufgelisteten Versionen dieses Dokumentes sind wie folgt zu verstehen:

- Die initiale Version jedes Dokuments ist v0.1
- Mit jeder Abgabe wird die Hauptversion des Dokuments um 1 erhöht und die Zwischenversion auf 0 zurückgesetzt (z.B. v1.2 → v2.0)
- Änderungen, ohne dass eine Abgabe stattfindet, führen zu einer Erhöhung um 1 in der Zwischenversion (z.B. v0.0 → v0.1)

Version	**Fertigstellung**	**Änderungen**	**Verantwortlich**
v1.0	16.05.2023	- Inhalte ergänzt - Dokument fertiggestellt - Architekturdiagramme hinzugefügt	Yan Wittmann Lauritz Fuchs Dominik Koschik
v0.1	27.04.2023	- Initiale Version des Dokuments angelegt - Dokumentstruktur angelegt	Lauritz Fuchs

Im vorliegenden Dokument nutzen wir die Begrifflichkeit des Kunden und etwaige Abwandlungen dessen repräsentativ für die sovanta AG.

1. Glossar

Begriff	Erklärung	Quelle
SAP BTP	Die SAP Business Technology Platform (SAP BTP) ist eine für SAP-Anwendungen in der Cloud optimierte Innovationsplattform. Sie vereint Funktionen für die Anwendungsentwicklung, das Datenmanagement und Analysen, Integration, Automatisierung und KI in einer zentralen Umgebung.	SAP BTP
RFID	RFID (Radio Frequency Identification) bezeichnet eine Technologie für Sender-Empfänger-Systeme zum automatischen und berührungslosen Identifizieren und Lokalisieren von Objekten und Lebewesen mit Radiowellen.	RFID
RFID-Tags	RFID-Tags sind eine Art Verfolgungssystem, das zur Identifizierung von Artikeln mithilfe einer Art smartem Barcode verwendet wird	RFID-Tag
RFID-Reader	Ein RFID-Reader ist ein Lesegerät, welches zum Lesen und/oder Beschreiben von RFID-Transpondern oder RFID-Tags verwendet wird und die Aufgabe hat verschiedene Objektinformationen, z.B. von Aufträgen, Werkstücken, Behälter, Material und Werkzeugen auszulesen.	RFID-Reader
API	Eine Programmierschnittstelle, kurz API genannt (von englisch application programming interface), ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird	API
JDBC	Java Database Connectivity (JDBC) ist eine Datenbankschnittstelle der Java-Plattform, die eine einheitliche Schnittstelle zu Datenbanken verschiedener Hersteller bietet und speziell auf relationale Datenbanken ausgerichtet ist.	JDBC
DevOps	Unter DevOps versteht man diverse Praktiken, Tools und eine Kulturphilosophie, die die Prozesse zwischen Softwareentwicklungs- und IT-Teams automatisieren und integrieren.	DevOps
CI/CD	CI/CD steht für Continuous Integration/Continuous Delivery. Es handelt sich hierbei um bewährte DevOps-Methoden zur Automatisierung in der Anwendungsentwicklung. Die Hauptkonzepte von CI/CD sind Continuous Integration (Kontinuierliche Integration), Continuous Delivery und Continuous Deployment (Kontinuierliche Verteilung).	CI/CD
CI/CD Pipeline	Eine CI/CD-Pipeline umfasst mehrere Schritte, die zur Bereitstellung einer neuen Softwareversion ausgeführt werden müssen. CI/CD-Pipelines verkörpern eine Praxis zur Verbesserung der Softwarebereitstellung.	CI/CD Pipeline

Cloud Foundry	Cloud Foundry ist eine Anwendungsplattform, die auf der SAP BTP zur Verfügung gestellt wird. Der Zweck von Cloud Foundry besteht darin, Entwicklern eine einfache Möglichkeit zu bieten, Anwendungen in der Cloud zu erstellen, bereitzustellen, auszuführen und zu skalieren, ohne sich um die zugrunde liegende Infrastruktur kümmern zu müssen.	Cloud Foundry
Virtual Machine Cluster	Als virtuelle Maschine (VM) wird in der Informatik die softwaretechnische Kapselung eines Rechnersystems innerhalb eines lauffähigen Rechnersystems bezeichnet. Cloud Foundry stellt mehrere VM's zur Ausführung von Anwendungen bereit, welche in dem Kontext als Cluster bezeichnet werden.	VM Cluster
Container	Container sind Softwarepakete, die alle Elemente enthalten, die zur Ausführung in beliebigen Umgebungen erforderlich sind. Container virtualisieren dadurch das Betriebssystem.	Container
REST	Eine REST-API (Representational State Transfer Application Programming Interface) ist eine Software-Schnittstelle, die auf dem REST-Architekturstil basiert. Sie ermöglicht die Kommunikation und den Austausch von Daten zwischen verschiedenen Anwendungen über das HTTP-Protokoll.	REST

2. Einführung und Ziele

2.1. Aufgabenstellung

Der Kunde möchte ein Produkt, welches er auf einer beliebigen Messe nutzen kann. Es soll für jeden Messebesucher, der den Stand unseres Kunden besucht, ein besonderes Erlebnis schaffen. Hierbei geht es vor allem darum, die SAP BTP zu präsentieren und sie für Interessenten als Entwicklungsplattform in Betracht zu ziehen. Ferner soll das Produkt zu weiteren Gesprächen mit Vertretern unseres Kunden am Messestand anregen und vermitteln, was die sovanta AG potenziellen Kunden bieten kann. Hierbei soll das Entwicklungskonzept der sovanta AG, die sovanta Innovation Factory, eine zentrale Rolle einnehmen.

2.2. Produktvision

Im Rahmen des Projekts möchten wir einen interaktiven Baukasten für Webapplikationen entwickeln, den Rocket Deployer. Dieser soll Anwender in die Lage versetzen, physische Blöcke, welche einzelne Services der SAP BTP oder die Funktionalität einer Applikation repräsentieren (bspw. ToDo-App), zu kombinieren und zu einer lauffähigen Webanwendung zusammenzusetzen. Jeder Block ist hierbei mit einem RFID-Tag versehen. Mithilfe von RFID-Readern kann jeder Block eingescannt und identifiziert werden. Anschließend werden auf einem Bildschirm Informationen über die ausgewählten Blöcke angezeigt. Wenn alle Bausteine ausgewählt wurden, welche für eine Webanwendung notwendig sind, kann diese generiert werden. Durch das Scannen eines QR-Codes kann die Webanwendung auf einem beliebigen mobilen Endgerät aufgerufen und verwendet werden. Mithilfe des Rocket Deployer möchten wir jedem Anwender ein einprägsames Erlebnis bieten, um die Ziele des Kunden zu erreichen.

2.3. Ziele der Applikation

Ziel der Webanwendung ist es, ein interaktives Erlebnis zu schaffen, welches die Besucher auf die SAP BTP aufmerksam macht. Eine intuitive und benutzerfreundliche Oberfläche soll den Besuchern hierbei die Vielseitigkeit der SAP BTP vermitteln und Ihre Funktionalitäten präsentieren.

Ebenso soll unser Produkt den Dialog zwischen Messebesuchern und Vertretern des Kunden fördern. Durch die Nutzung des Rocket Deployer sollen Besucher angeregt werden, sich intensiver mit den Möglichkeiten der SAP BTP und den Diensten des Kunden auseinanderzusetzen.

2.4. Stakeholder

2.4.1. Interne Stakeholder

Rolle	**Konkrete Vertreter**	**Beschreibung / Funktion**	**Projekt-relevanz**
Projektmanager	• Wolfgang Schramm • Peter Knauber	• Überwacht den Projektfortschritt • Beurteilt und bewertet das Projektteam und die Produktinkremente • Stellt Fachwissen zur Verfügung und gibt regelmäßig Feedback • Legt Deadlines für Dokumente und Produktinkremente fest	Hoch
Unterstützung	• Tutoren	• Bietet konstruktives Feedback • Unterstützt den Prozess durch das Bereitstellen von Wissen und Erfahrung	Mittel
Projektteam	• Dominik Koschik • Eddi Bludau • Julian Komarek • Jonas Fügen • Lauritz Fuchs • Sophie Humbert • Yan Wittmann	• Zuständig für die Entwicklung und das Design des Produkts • Definition und Spezifikation von Anforderungen • Erstellung von Dokumenten und Qualitätssicherung • Projektplanung und Kommunikation mit dem Auftraggeber	Hoch

2.4.2. Externe Stakeholder

Rolle	**Konkrete Vertreter**	**Beschreibung / Funktion**	**Projekt-relevanz**
Messebesucher		• Benutzt Endprodukt • Soll durch Applikation Interesse an der SAP BTP und der sovanta AG gewinnen	Mittel
Auftraggeber (sovanta AG)	• Jakob Frankenbach • Larissa Haas • Alina Meiseberg • Thomas Bechberger • Louise Hebestreit	• Entscheidungsträger über Richtung des Produkts • Definition und Validierung von Anforderungen und Zielen • Überwachung des Entwicklungsprozesses und des Budgets • Sicherstellung der Einhaltung von Standards	Hoch

3. Randbedingungen

Der Ausdruck Pflicht und Abwandlungen dessen, implizieren im vorliegenden Kapitel, dass die Nichterfüllung einer Pflicht zu einer negativen Beeinträchtigung einer Bewertung führen kann und ist nicht in rechtlichem Kontext zu betrachten.

3.1. Technische Randbedingungen

SAP BTP als Zielplattform

Alle Komponenten der Endanwendung sollen mithilfe der SAP BTP umgesetzt oder auf der SAP BTP bereitgestellt werden.

Einschränkung der erlaubten Open-Source Lizenzen

Der Kunde schreibt als verwendbare Open-Source Lizenzen die Folgenden vor: BSD, Apache, MIT.

3.2. Finanzielle Randbedingungen

Eingeschränkte Ressourcenbereitstellung durch den Kunden

Das Projekt findet im Rahmen einer Projektsimulation statt. Der Kunde stellt kein pauschales Projektbudget (in Euro) zur Verfügung. Die Bereitstellung bzw. minimale Ausweitung verfügbarer Software - Ressourcen auf der SAP BTP muss mit dem Kunden verhandelt werden.

Die bereitgestellten Ressourcen liegen extrem weit unterhalb dessen eines realen Softwareprojekts mit ähnlichem Umfang. Der Entwicklungsprozess kann hierdurch verlangsamt oder eingeschränkt werden

3.3. Organisatorische Randbedingungen

Eingeschränkte Verfügbarkeit innerhalb des Entwicklerteams

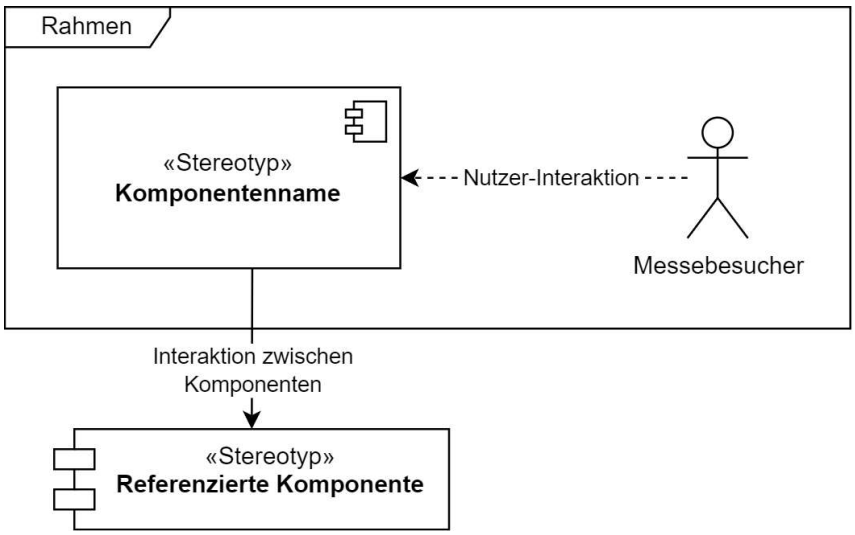
Keines der Teammitglieder von Team Leap ist vertraglich an das Produkt bzw. dessen Entwicklungsprozess gebunden. Somit hat jeder Entwickler die Freiheit, das Team jederzeit zu verlassen oder auszutreten. Hierbei kann kein adäquater Ersatz bereitgestellt werden. Ebenso kann jedes Teammitglied seine Arbeitszeit selbst bestimmen.

Vorgegebenes Vorgehensmodell - Eingeschränkte Planungsfreiheit

Die Nutzung von Scrum, zumindest die Durchführung von Sprints, wird durch die Professoren verpflichtend vorgegeben. Ebenso wird die Sprintlänge von einer Woche vorgegeben. Sprintstarts, sowie deren Ende, sind an feste Daten gebunden. Des Weiteren müssen verschiedene Dokumente an Fristen eingereicht werden, welche vor Projektstart durch die Professoren bestimmt wurden. Frist Verschiebungen müssen hierbei individuell verhandelt werden. Ferner sind Termine für Fachenglisch und Teamentwicklung-Workshops vorgeschrieben und verpflichtend wahrzunehmen.

4. Kontextabgrenzung

In den Kontext-, Baustein- und Verteilungssicht-Diagrammen, die in diesem Dokument zu finden sind, werden die folgenden Symbole und Pfeile verwendet:



4.1. Fachlicher Kontext

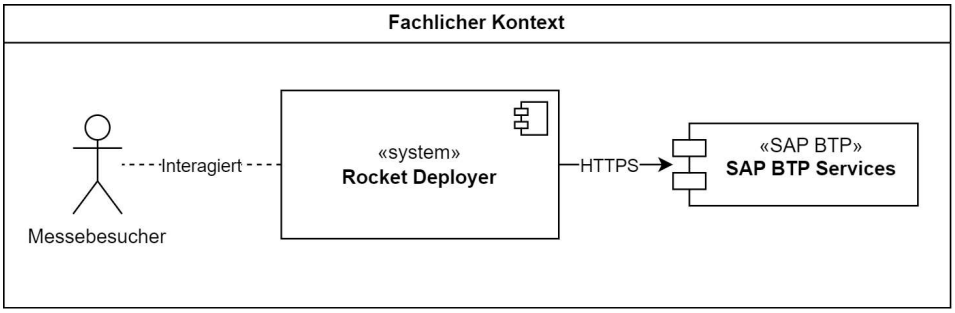


Abb. 1: Fachlicher Kontext

4.1.1. Rocket Deployer

Der Rocket Deployer besteht aus eigens entwickelten Soft - und Hardwarekomponenten. Er beinhaltet alle User Interfaces sowie die Anwendungslogik.

4.1.2. Messebesucher

Der Messebesucher interagiert auf folgende Art und Weise mit dem Rocket Deployer: Zum einen wählt er Blöcke aus und legt sie in eine Vorrichtung auf die entsprechenden Stellen. Andererseits bekommt er Informationen und eventuelle Anweisungen über einen Monitor grafisch und textuell bereitgestellt.

Nachdem der Anwender eine Anwendung zusammengestellt hat und diese erstellt wurde, kann er sie auf seinem mobilen Endgerät aufrufen und benutzen.

4.1.3. SAP BTP Services

Software Services, welche auf der SAP BTP bereitgestellt werden, interagieren folgendermaßen mit dem Rocket Deployer:

Instanzen der verwendeten SAP Services laufen auf der SAP BTP und bieten jeweils eine Schnittstelle, sodass der Rocket Deployer mit den Services kommunizieren und deren Funktionalität nutzen kann.

4.2. Technischer Kontext

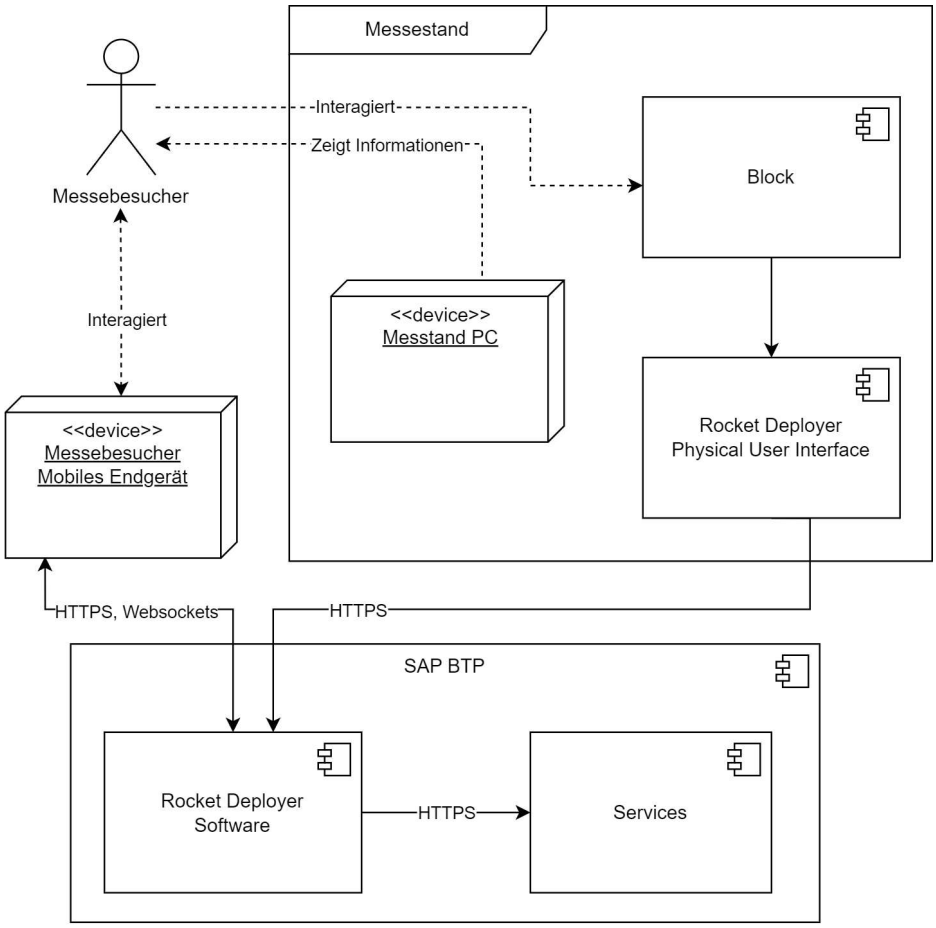


Abb. 2: Technischer Kontext

4.2.1. Messestand PC

Der Messestand PC dient zur Anzeige von Informationen über den letzten Block, welcher auf einem RFID Reader gelegt wurde. Außerdem zeigt er, wo sich der Besucher aktuell im Prozess befindet und welche Schritte als nächstes genommen werden können.

4.2.2. Raspberry Pi

Der Raspberry Pi scannt mit RFID Readern die Baublöcke und leitet diese mittels eines HTTPS-Requests an das Backend weiter. Wenn der Besucher seine gewünschte Konfiguration gewählt hat, dann kann er auf den Startknopf drücken und seine gewünschte Applikation wird fertiggestellt.

4.2.3. Blöcke

Die Blöcke haben einen RFID-Tag mit einer festen ID an ihrer Unterseite, die zur Identifizierung des Blocks gegenüber des RFID-Readers dient.

4.2.4. SAP BTP

Die SAP BTP hostet unseren Rocket Deployer und stellt die Services für die Baublöcke zur Verfügung.

4.2.5. Messebesucher Mobiles Endgerät

Der Messebesucher kann seine fertig erstellte Applikation mittels eines QR-Codes auf seinem Mobilgerät scannen und während der Messe weiterverwenden.

5. Lösungsstrategie

5.1. Technologieentscheidungen

5.1.1. PostgreSQL

Für unsere Anwendung haben wir uns für das relationale Datenbankverwaltungssystem (RDBMS) PostgreSQL entschieden. Die Auswahl dieses Systems basiert auf mehreren Schlüsselfaktoren:

- PostgreSQL ist ein nativer Service auf der SAP BTP, was eine nahtlose Integration mit Anwendungen ermöglicht, die auf der SAP BTP gehostet werden. Dies erleichtert die Anwendungsentwicklung und minimiert die Komplexität des Systemmanagements.
- Wir verfügen bereits über Kenntnisse und Erfahrungen mit PostgreSQL, was die Einarbeitungszeit erheblich verkürzt und eine effiziente Integration in unsere Anwendung sicherstellt.
- PostgreSQL ist ein Open-Source-RDBMS. Dies bedeutet, dass es keine Lizenzkosten verursacht und eine große Community hat, welche Wissen zur freien Verfügung stellt.

  Betrachtete Alternativen:

  SAP HANA

  SAP HANA ist eine In-Memory-Datenbank, die für hohe Leistungsfähigkeit und schnelle Zugriffe auf große Datenmengen konzipiert ist. Obwohl sie als Service auf

der SAP BTP verfügbar ist und eine einfache Integration ermöglicht, entschieden wir uns gegen SAP HANA aus folgenden Gründen:

- Unsere Anwendung hat keinen hohen Datenbedarf, da sie hauptsächlich Anwendungskonfigurationen der Besucher speichert. Daher ist die hohe Leistungsfähigkeit von SAP HANA für unseren Anwendungsfall nicht notwendig.
- Im Vergleich zu anderen Datenbankalternativen sind die Kosten für SAP HANA signifikant höher, unter anderem aufgrund der Lizenzkosten, die anfallen würden. Die sovanta AG ist nicht bereit, diese zu tragen.

Datenbanksysteme, die nicht auf der BTP zur Verfügung stehen

Es gibt heute ein breites Spektrum an Datenbanksystemen, die alle Vor- und Nachteile bieten. Diese brauchen allerdings aus einem Grund nicht betrachtet werden:

- Die sovanta AG möchte, dass Services der SAP BTP für die Datenhaltung verwendet werden.

5.1.2. Angular

Unsere Wahl für das Frontend der Anwendung ist das Framework Angular.

Angular ist ein JavaScript-Framework zur Entwicklung von Single-Page-Anwendungen (SPA) zur Erstellung interaktiver Benutzeroberflächen.

Die Entscheidung für Angular basiert auf diesen Faktoren:

- Aufgrund der hohen Komplexität unserer Anwendung haben wir uns grundsätzlich für die Verwendung eines Frameworks entschieden, um den Entwicklungsaufwand und die Komplexität des Quellcodes so gering wie möglich zu halten.
- Aufgrund bestehender Angular-Kenntnisse kann die Zeit für die Einarbeitung gering gehalten werden.
- Der modulare Aufbau von Angular-Anwendungen bietet uns die Möglichkeit, unser stark modularisiertes Anwendungskonzept gut zu modellieren und zu strukturieren.
- Die Verwendung von TypeScript in Angular ermöglicht Typsicherheit und die Vermeidung von potenziellen Fehlern bereits in der Kompilierung.

Betrachtete Alternativen:

ReactJS & VueJS

ReactJS sowie Vue.js sind JavaScript-Bibliotheken zur Entwicklung von Benutzeroberflächen.

- Zwar gelten ReactJS und Vue.js als leichtgewichtiger im Vergleich zu Angular, da es sich nicht um vollwertige Frameworks handelt, jedoch bietet uns der größere Funktionsumfang (siehe folgende Punkte) von Angular Vorteile bei der Umsetzung unseres Konzepts.

- Das eingebaute Routing von Angular ermöglicht die einfache Handhabung vieler verschiedener Seiten, was für die Umsetzung unseres Konzepts mit vielen verschiedenen Seiten von Vorteil ist. ReactJS und Vue.js würden an dieser Stelle weitere Bibliotheken benötigen.
- Angular bietet einen eingebauten HTTP-Client für die REST-Kommunikation, diese findet Anwendung in unserer Applikation. ReactJS und VueJS würden auch hier weitere Bibliotheken benötigen.

5.1.3. Spring Boot

Unsere Wahl für das Backend der Anwendung ist das Framework Spring Boot.

Spring Boot ist ein auf Java basierendes Framework, das die Erstellung eigenständiger, Anwendungen erleichtert. Es wurde entwickelt, um die Komplexität der Entwicklung und Konfiguration von Spring-Anwendungen zu reduzieren.

Die Entscheidung für Spring Boot basiert auf folgenden Faktoren:

- Aufgrund seiner weiten Verbreitung hat Spring Boot eine große und aktive Community. Dies führt zu einer Fülle von Ressourcen, einschließlich Tutorials, Beispielen und Lösungen für gängige Probleme.
- Wegen der fundierten Java-Kenntnisse der Teammitglieder war ein Java Backend eine einfache Entscheidung: Dies reduziert die Einarbeitungszeit in Programmiersprachen und deren Frameworks.
- Spring Boot ermöglicht eine einfache Konfiguration mit geringer Komplexität.
- Spring Boot ist vollständig kompatibel mit der SAP BTP. Beispielsweise hebt das Standard-Buildpack auf der BTP viele der Konfigurationsparameter automatisch auf und konfiguriert die Umgebung automatisch.

Betrachtete Alternativen:

Java EE (Enterprise Edition)

Java EE ist eine weit verbreitete Plattform für die Entwicklung von Unternehmensanwendungen. Wir haben uns jedoch aus folgenden Gründen dagegen entschieden:

- Komplexität und Entwicklungsgeschwindigkeit: Java EE kann komplexer und langsamer in der Entwicklung sein als Spring Boot, insbesondere für kleinere Projekte wie unseres. Der Overhead, der benötigt wird, um beispielsweise einfache Endpunkte anzulegen, ist für unser Projekt nicht gerechtfertigt.
- Flexibilität: Java EE kann in Bezug auf die Konfiguration und Integration von Drittanbieter-Bibliotheken weniger flexibel sein als Spring Boot.
- Java EE kann im Vergleich zu Spring Boot mehr Systemressourcen beanspruchen, da es ein vollständiges Enterprise-Framework ist. Für unser Projekt, das eher leichtgewichtige und effiziente Lösungen benötigt, kann dies zu unnötigem Overhead führen.

Quarkus

Quarkus ist ein Kubernetes-native Java-Framework, das für Java Virtual Machines (JVMs) und native Kompilierung konzipiert ist, was zu schnelleren Startzeiten und geringerem Speicherverbrauch führt. Wir haben uns aus folgenden Gründen dagegen entschieden:

- Reife, Unterstützung und Lernkurve: Quarkus ist ein relativ neues Framework mit entsprechend weniger Online-Ressourcen und einer steileren Lernkurve im Vergleich zu Spring Boot.
- Quarkus ist stark auf den Einsatz mit Kubernetes ausgerichtet. Da wir jedoch auf der SAP Business Technology Platform (BTP) bereitstellen und nicht mit Kubernetes arbeiten, wäre ein Großteil der spezifischen Vorteile von Quarkus in unserem Kontext nicht relevant.
- Da Quarkus noch relativ neu ist, gibt es noch nicht die nahtlose Integration und Unterstützung innerhalb der SAP BTP wie bei Spring Boot.

5.1.4. Maven

Als Build-Tool für das Java-Backend und zur Verwaltung unserer Projektabhängigkeiten haben wir uns für Apache Maven entschieden.

Maven ist ein weit verbreitetes Tool, das auf dem Konzept eines Projektobjektmodells (POM) basiert und kann Projekte in verschiedenen Programmiersprachen, vor allem jedoch in Java, bauen. Es ermöglicht einen einheitlichen Build-Prozess und bietet eine zentrale Informationsdarstellung.

Wir haben uns für Maven aus folgenden Gründen entschieden:

- Einfache Konfiguration: Durch die Verwendung einer zentralen XML-Datei ist die Konfiguration mit Maven einfach und übersichtlich.
- Leistungsfähige Abhängigkeitsverwaltung: Maven bietet eine robuste und automatische Verwaltung von Bibliotheksabhängigkeiten, einschließlich der Auflösung und Aktualisierung.
- Breite Unterstützung und große Community: Maven wird von einer breiten Palette von integrierten Entwicklungsumgebungen und kontinuierlichen Integrationsservern unterstützt. Es hat eine große, aktive Community, die ständig Verbesserungen vornimmt und bei Problemen helfen kann.
- Erfahrung im Team: Eines der Teammitglieder hat bereits große Erfahrung mit Maven, was die Lernkurve abflacht und die Produktivität erhöht.

Betrachtete Alternative:

Gradle

Gradle ist ein weiteres leistungsfähiges Build-Tool, das insbesondere in den letzten Jahren an Beliebtheit gewonnen hat. Es bietet eine flexible und leistungsfähige Plattform für das Build-Management und wird in Projekten wie Android Studio verwendet.

Wir haben uns jedoch aus folgenden Gründen gegen Gradle entschieden:

- Komplexität: Gradle kann wegen seiner Flexibilität und seines funktionalen Aufbaus komplexer sein als Maven. Es kann mehr Zeit und Aufwand erfordern, um die Gradle-Scripts zu verstehen und anzupassen.
- Unser Team hat mehr Erfahrung in der Anwendung von Maven. Eine Entscheidung für Gradle würde mehr Einarbeitungszeit erfordern.

5.1.5. Websockets

Für unsere Anwendung nutzen wir Websockets zur Ermöglichung von bidirektionaler Kommunikation zwischen dem Server und dem Client. Der Client zum einen die Frontend-UI auf dem Hauptmonitor und zum anderen die Applikationen, die der Messebesucher zusammenstellt und auf seinem Handy verwendet. Websockets bieten einen ständigen, offenen Kommunikationskanal, der es dem Server ermöglicht, Daten an den Client zu senden, sobald sie verfügbar sind.

Die Entscheidung für Websockets basiert auf folgenden Faktoren:

- Bidirektionale Kommunikation: Unsere Anwendung benötigt eine Echtzeitkommunikation zwischen Server und Client, vor allem von den Applikationen, die durch die Messebesucher zusammengestellt werden. Websockets ermöglichen bidirektionale Nachrichtenübertragung ohne Polling, was für eine reaktive Benutzeroberfläche in unserer Anwendung unerlässlich ist.
- Leistung: Websockets vermeiden Overhead, der durch wiederholte HTTP-Anfragen und -Antworten entsteht. Sobald der Verbindungsaufbau hergestellt ist, können Daten mit sehr geringer Latenz übertragen werden.
- Standardisierung: Websockets sind ein standardisiertes Protokoll, das breite Unterstützung in modernen Webbrowsern und Netzwerkhardware hat. Damit ist es für uns einfach, diese in unsere Applikation zu integrieren.
- Spring Boot unterstützt Websockets nativ, was die Anbindung an das Backend stark vereinfacht.

Betrachtete Alternativen:

Polling und Long Polling

Polling und Long Polling sind Techniken, die vor allem in der Vergangenheit verwendet wurden, um Echtzeitkommunikation in Webanwendungen zu ermöglichen. Dabei fragt der Client regelmäßig den Server nach neuen Daten. Wir haben uns jedoch aus folgenden Gründen gegen diese Techniken entschieden:

- Performance und Effizienz: Sowohl Polling als auch Long Polling können zu einem hohen Netzwerk- und Ressourcenverbrauch führen, da sie viele Anfragen erzeugen.
- Latenz: Bei diesen Techniken kann es zu einer Verzögerung kommen, bevor der Client neue Daten erhält. Dies ist insbesondere bei Polling der Fall, da die Frequenz der Anfragen festgelegt ist.
- Komplexität: Die Implementierung von Polling und primär Long Polling kann komplex sein und eine sorgfältige Handhabung erfordern, um Probleme wie Anfragen, die in der falschen Reihenfolge ankommen, zu vermeiden.

Server-Sent Events (SSE)

Server-Sent Events sind eine Technologie, die es einem Server ermöglicht, automatisch Daten an den Client zu senden, sobald neue Daten verfügbar sind. Obwohl sie eine unidirektionale Kommunikation vom Server zum Client ermöglichen, entschieden wir uns gegen SSE aus folgenden Gründen:

- Browser-Kompatibilität: Obwohl SSE in den meisten modernen Browsern gut unterstützt wird, gibt es einige Kompatibilitätsprobleme, vorwiegend mit älteren Versionen von Internet Explorer und bestimmten mobilen Browsern. Dies könnte die Zugänglichkeit und Benutzererfahrung unserer Anwendung einschränken.
- Verbindungslimit: SSE hat ein Limit für gleichzeitige offene Verbindungen, das bei einigen Browsern auf sechs pro Domain begrenzt ist. Bei einer großen Anzahl von Benutzern oder bei Anwendungen, die mehrere Verbindungen pro Benutzer erfordern, könnte dies ein Problem darstellen.

5.2. Architekturmuster

5.2.1. Client-Server

In unserem Projekt setzen wir das Client-Server-Designmuster durch die Nutzung von Angular für das Frontend und Java für das Backend um. In dieser Architektur agiert das Angular-Frontend als der Client, der Anfragen an das Java-Backend, den Server, sendet und von diesem Antworten empfängt.

Die Kommunikation zwischen Client und Server erfolgt durch definierte Endpunkte, über die Daten empfangen oder übermittelt werden können. Das Frontend ist dabei für die Präsentation der Daten und die Interaktion mit dem Benutzer zuständig, während das Backend die Geschäftslogik und Datenverarbeitung übernimmt.

Diese klare Trennung von Präsentation und Datenverarbeitung trägt zu einer skalierbaren und wartbaren Architektur bei. Sie ermöglicht es uns, das Frontend und das Backend unabhängig voneinander zu entwickeln und zu verbessern, was sowohl die Produktivität als auch die Qualität unserer Software steigert.

5.2.2. Data Access Object (DAO)

Allgemein kapselt das DAO Muster Zugriffe auf eine Datenquelle zur Speicherung und Wiederherstellung von Daten. Es stellt eine Schnittstelle zur Verfügung, die von der restlichen Anwendung genutzt wird, um auf die zugrundeliegenden Daten zuzugreifen. Die konkrete Implementierung dieser Schnittstelle kann dann die tatsächlichen Datenbankaufrufe durchführen.

Im Java Backend unserer Applikation ist der Zugriff auf eine Datenbank ein immer wiederkehrendes Thema. Unsere Applikation muss in verschiedenen Konfigurationen mit unterschiedlichen Datenbanken lauffähig sein:

- Lokal mit einer selbst-gehosteten PostgreSQL Instanz
- Auf der SAP BTP mit der PostgreSQL Datenbank als BTP Service
- In den JUnit Testfällen mit einer Embedded Test-Datenbank

Daher ist es wichtig, den Code, der für diesen Zugriff verantwortlich ist, vom Rest der Anwendung zu trennen und ein Interface zu bieten, über das Verbindungen aufgebaut und SQL-Abfragen getätigt werden können.

Hierbei gibt es bei unserer Anwendung zwei Abstraktionsebenen, die aufeinander aufsetzen. Die erste erkennt, in welcher Umgebung und mit welcher Konfiguration die Applikation läuft und erstellt mittels des Singleton-Patterns eine Instanz, mit der allgemein auf die ausgewählte Datenbank zugegriffen werden kann. Diese Zugriffe erfolgen in jedem Fall über das JDBC-Framework.
Die zweite Ebene setzt darauf auf und ähnelt dem Repository-Konzept, das von Spring verwendet wird, um auf einzelne Tabelleneinträge in Datenbanken zuzugreifen. Diese Klassen sind bei unserem Projekt die JdbcTable und JdbcRow. Sie erlauben es, ein Schema für Tabellen zu definieren und einheitliche Abfragen und Änderungen an diesen Daten auf der Datenbank durchzuführen.

5.2.3. Model-View-Controller

Das Model-View-Controller (MVC) Muster ist ein Designprinzip, das eine klare Trennung zwischen Daten (Model), Darstellung (View) und Logik (Controller) einer Anwendung ermöglicht. Es stellt ein effizientes Paradigma für die Entwicklung von Software dar, insbesondere für Benutzeroberflächen, da es die Wiederverwendbarkeit und Parallelentwicklung fördert.

In unserem Projekt setzen wir das MVC-Muster durch die Verwendung des Angular-Frameworks um. Dies erleichtert nicht nur die Wartung und Erweiterung des Systems, sondern ermöglicht es auch mehreren Entwicklern, parallel an verschiedenen Komponenten des Systems zu arbeiten. Darüber hinaus verbessert die klare Abgrenzung zwischen „Model“, „View“ und „Controller“ die Testbarkeit des Systems, da einzelne Komponenten isoliert getestet werden können.

Das Model repräsentiert dabei beliebige Daten (in Angular Typescript, JSON Dateien), die View (Angular HTML Dateien, Templates) stellt die Benutzeroberfläche dar und der Controller (Angular Typescript Dateien) verarbeitet die Eingabe und steuert die Interaktion zwischen dem Model und der View.

5.3. Qualitätssicherung

5.3.1. Branching-Strategie (git)

Wir verwenden drei Arten von Branches in unserem Projekt:

1. **Feature Branches**: Diese werden für die Entwicklung neuer Funktionen oder das Beheben von Bugs verwendet. Jeder Feature Branch basiert auf dem dev Branch. Nach der Fertigstellung des Features oder dem Beheben eines Bugs wird der Feature-Branch in den dev Branch gemerged.
   Die Branch-Bezeichner referenzieren immer ein entsprechendes Ticket auf unserem Jira Board und haben die Form LEAP-<Ticketnummer>-<Ticket Name>.
2. dev **Branch**: Dieser Branch enthält den aktuellen Entwicklungsstand. Wenn die Änderungen in einem Feature Branch abgeschlossen, reviewed und getestet sind, werden sie in den dev Branch gemerged. Sobald das passiert, wird eine Version der Applikation automatisch durch unsere Pipeline gebaut und als Development-Applikation auf unseren Cloud Foundry Space hochgeladen.
3. prod **Branch**: Dieser Branch beinhaltet eine stabile Version unserer Anwendung. Wenn die Änderungen im dev Branch stabil sind, werden sie in den prod Branch gemerged. Eine Version der Applikation wird automatisch durch unsere Pipeline gebaut und als Production-Applikation auf unseren Cloud Foundry Space hochgeladen.

5.3.2. Kontinuierliche Integration und Deployment (CI/CD)

Unsere CI/CD-Pipeline wird durch GitLab gesteuert und beinhaltet die folgenden Stufen:

1. **Build**: In dieser Phase wird der Code des Backends und des Frontends kompiliert und gebaut. Wir verwenden unterschiedliche Build-Skripte für unsere dev und prod

Branches, die entsprechende Endpoints und Konfigurationen in die Applikationen injizieren. Dieser Prozessschritt wird ausgelöst, sobald ein Merge-Request geöffnet wird oder bei einem neuen Push auf einen bereits geöffneten Merge-Request.
2. **Deploy**: Wenn alle Tests erfolgreich durchlaufen wurden, wird die gebaute Anwendung auf die entsprechende Umgebung (Entwicklung oder Produktion) deployt. Dies wird nur dann ausgeführt, wenn der Merge-Request tatsächlich durchgeführt wurde, sodass darauf immer ein lauffähiger Stand vorhanden ist.

Diese Pipeline stellt sicher, dass jede Code-Änderung sofort getestet und bei gemergten Änderungen in die entsprechende Umgebung deployt wird. Bei gescheiterten Build-Pipelines bekommt der entsprechende Commit-Autor eine E-Mail, die ihn darüber informiert. Durch diese Automatisierung können wir Fehler schnell erkennen und beheben und die Qualität unserer Software sicherstellen

Standardmäßig bietet GitLab nur ein Kontingent von 400 Minuten pro Monat für die Runner, was für unseren Use-Case bei Build-Zeiten von ca. 3-4 Minuten nicht ausreicht. Darum haben wir auf einer Free-Tier EC2 Instanz (Amazon Web Services) einen Custom GitLab Runner installiert und konfiguriert, der unsere Pipelines ausführen kann.

5.3.3. Code-Reviews

Jede Änderung des Quellcodes, welche in den dev oder prod Branch gemerged werden soll, muss einen Code-Review-Prozess durchlaufen. Dabei wird der Code von mindestens zwei Teammitgliedern geprüft, welche nicht an dessen Entwicklung beteiligt waren. Dies erlaubt es uns, Fehler, potenzielle Probleme und Verbesserungsmöglichkeiten zu identifizieren, bevor die Änderungen übernommen werden.

Bei diesem Review nutzen wir das GitLab Feature, Änderungen des Merge-Requests als “Viewed” zu markieren. Alle Änderungen müssen als “Viewed” markiert sein, bevor ein Merge durchgeführt werden kann.

5.3.4. Teststrategie

Im Java Backend verwenden wir JUnit, ein weit verbreitetes Framework für Unit-Tests in Java.

Im Angular Frontend setzen wir das Framework Jest ein, ein Behavior-Driven Development (BDD) Framework für JavaScript. Jest wird verwendet, um verschiedene Aspekte der Frontend-Logik zu testen.

Die genauen Details der Teststrategie werden in einem separaten Dokument genauer beschrieben.

6. Bausteinsicht

6.1. Ebene 1

Der Rocket Deployer ist eine Lösung, die sowohl physische als auch softwarebasierte Komponenten umfasst. Die oberste Ebene (1) des Rocket Deployer Systems ist in zwei Hauptkomponenten unterteilt: die physische Benutzerschnittstelle und die Software, die den Applikationsstatus verwaltet und das User Interface (UI) bereitstellt.

Der Anwender interagiert mit beiden System:

- Er nimmt die physischen Blöcke und legt sie auf die RFID-Reader
- Mittels eines QR Codes kann er die erstellte Webanwendung öffnen
- Im Browser seines Handys kann er die Webanwendung nutzen

Zudem greift das Backend in der Software-Komponente auf diverse SAP-BTP-Services zu. Diese werden in Ebene 2 weiter aufgeschlüsselt.

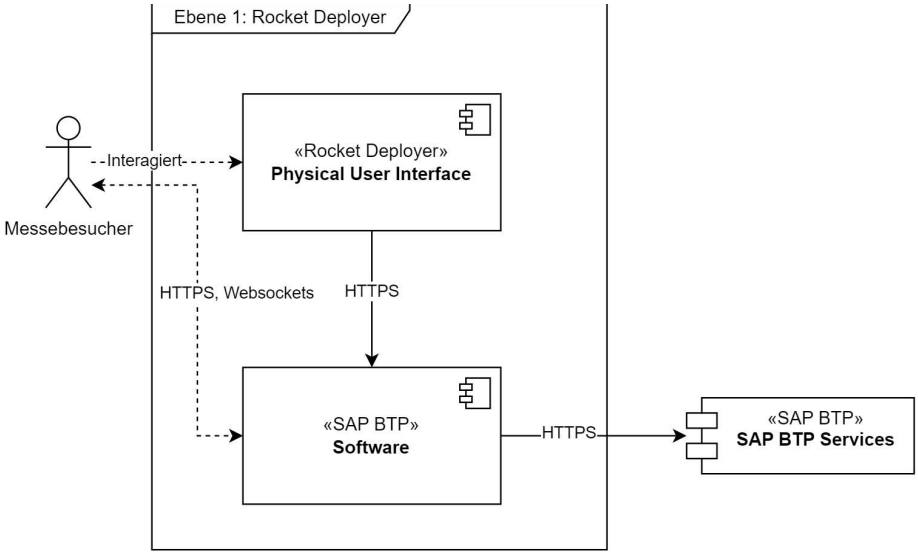


Abb. 3: Bausteinsicht Ebene 1: Rocket Deployer

6.2. Ebene 2

Im ersten Diagramm werden sowohl das Physical User Interface, als auch unsere Software auf der SAP BTP aufgeschlüsselt.

Als Kommunikationsprotokoll zwischen den einzelnen Komponenten wird HTTPS verwendet. Intern verwendet die SAP BTP einen App Router, um über die Domain und den Host zu entscheiden, an welche Applikation die Requests geleitet werden sollen. Ebenso kommunizieren die beiden Applikationen miteinander, sie machen Requests, die dann vom App-Router an die entsprechende Applikation geleitet werden.

Physical User Interface

Die physische Benutzerschnittstelle, mit dem der Messebesucher interagiert, besteht aus den Applikationsblöcken, die der Besucher auf die RFID-Reader des **RFID Read & Launch System** legt. Jeder dieser Blöcke enthält einen RFID-Tag, der sich mittels eines eindeutigen Identifiers gegenüber dem RFID-Reader identifiziert, wenn der Block auf den Reader gelegt wird. Dieses Signal wird dann vom Raspberry Pi, der an die RFID-Reader angeschlossen ist, in eine lesbare Information umgewandelt und über das auf dem Raspberry Pi laufenden Python-Programm an das Java-Backend über einen HTTPS-Request gesendet.

Zusätzlich zum RFID Read & Launch System besteht die physische Benutzerschnittstelle aus einem Deploy-Knopf. Nachdem der Besucher seine Auswahl an Blöcken getroffen hat, kann er den „Deploy"-Knopf drücken, um seine Applikation zu starten. Dieser Knopfdruck wird ebenfalls vom Raspberry Pi erfasst und an das Java-Backend gesendet.

SAP BTP: Software

Die Softwarekomponente des Systems, die auf der SAP BTP läuft, besteht aus mehreren Teilen. Das Java-Backend ist das Herzstück der Software und verwaltet den Zustand der Applikation. Es nimmt die Informationen, die vom Raspberry Pi kommen, entgegen und aktualisiert entsprechend den Zustand der Applikation. Es kommuniziert auch mit dem PostgreSQL-Dienst, um die notwendigen Daten zu speichern und zu verwalten.

Das Java-Backend bietet auch Endpunkte für die beiden Angular Frontends. Das Haupt-Angular-Frontend wird auf einem Monitor am Messestand angezeigt und zeigt den aktuellen Zustand der Applikation an. Es interagiert mit dem Java-Backend über Websockets, um Echtzeit-Updates zu erhalten. Das andere Angular-Frontend bietet die Benutzerschnittstelle für die individuellen Applikationen, die von Messebesuchern erstellt werden. Jede dieser Applikationen hat eine einzigartige ID und wird durch diese ID unterschiedlich gerendert.

Darüber hinaus nutzt das Java-Backend die Umgebungsvariable VCAP_SERVICES, um auf den PostgreSQL- und andere Dienste zuzugreifen, die auf der SAP BTP gehostet werden.

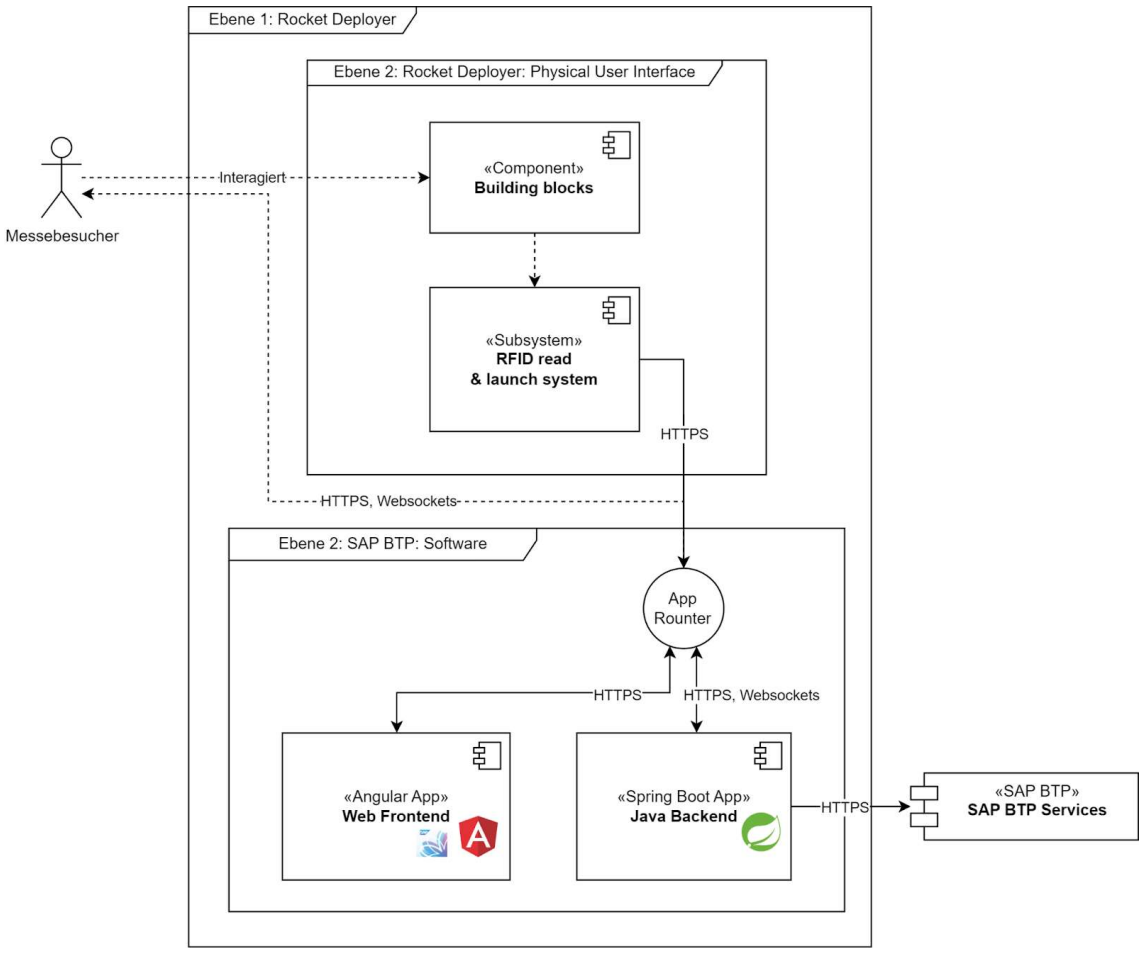


Abb. 4: Bausteinsicht Ebene 2: Rocket Deployer

SAP BTP: Services

In der nachfolgenden Grafik sind die Services aufgeführt, die von der SAP BTP bereitgestellt und von unserer Applikation genutzt werden.

Der Service Manager der SAP BTP ist dafür verantwortlich, die Umgebungsvariable VCAP_SERVICES zu setzen und an das Java Backend zu übergeben. Dies ermöglicht die Authentifizierung gegenüber der PostgreSQL-Datenbank und anderen Services.

Diese umfassen:

- PostgreSQL Database:
  Details zu PostgreSQL und warum wir es nutzen, wurden bereits im Kapitel „Technologieentscheidungen“ erläutert.
- xsuaa Authentication Service:
  XSUAA steht für "Extended Services for User Account and Authentication". Es handelt sich hierbei um einen zentralen Bestandteil der Sicherheitsinfrastruktur der SAP Cloud Plattform. Dieser Service ist verantwortlich für die Authentifizierung und

Autorisierung von Benutzern, die auf die erstellten Anwendungen zugreifen. Er gewährleistet, dass nur berechtigte Benutzer auf die entsprechenden Anwendungen zugreifen können. In unserem Fall verwenden wir diesen Service für das optionale Login-Fenster, das durch einen der Bausteine repräsentiert wird, um Benutzerkonten zu erstellen.

- SAP Conversational AI:
  Die SAP Conversational AI ermöglicht es, einen eigenen Chatbot zu trainieren, der auf bestimmte Eingabeaufforderungen (Prompts) reagieren kann. Wir verwenden diese Funktion, um dem Messebesucher die von ihm genutzte Applikation zu erläutern.
- AI Service:
  Dieser Service bezieht sich auf eine zum jetzigen Zeitpunkt noch zu definierende KI-Funktion, die von der SAP BTP bereitgestellt wird.

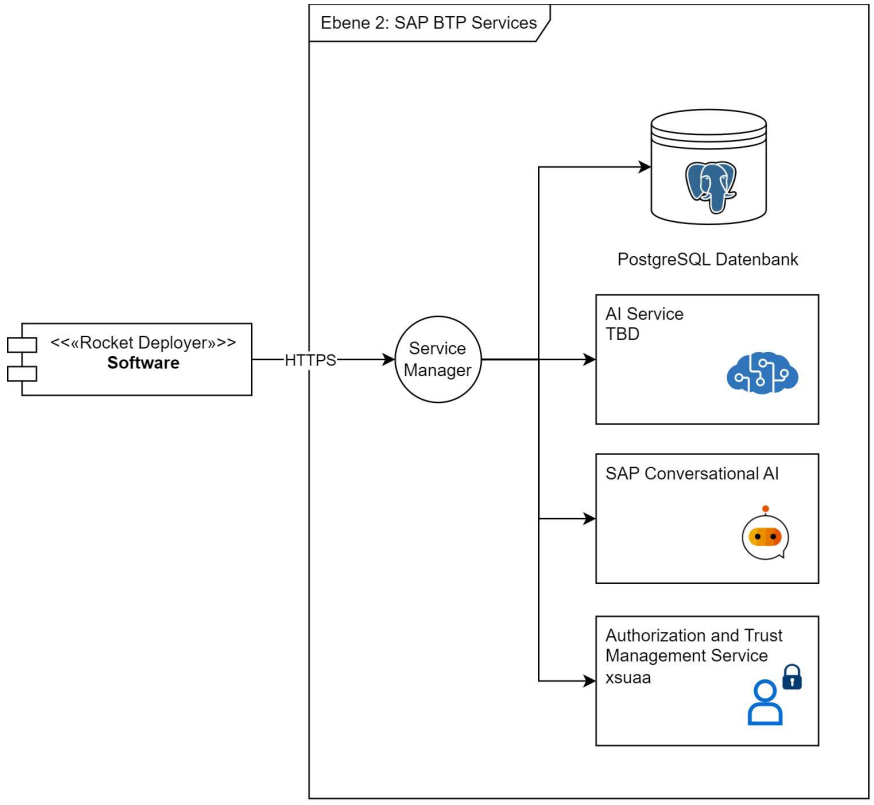


Abb. 5: Bausteinsicht Ebene 2: SAP BTP Services

6.3. Ebene 3

Diese Ebene enthält die spezifischen Hardwarekomponenten der physischen Benutzerschnittstelle und die Endpunkte sowie die Struktur der Softwarekomponenten.

Physisches User Interface: Baublöcke

Hierbei handelt es sich um die Bausteine der Applikation, mit denen der Besucher seine Applikation zusammenstellt. Sie bestehen aus einem bisher nicht festgelegten Material und sind an der Unterseite mit einem RFID-Tag versehen, der eine eindeutige ID trägt und an die RFID-Leser weitergeben kann. Die drei verschiedenen Kategorien der Blöcke sind visuell unterscheidbar durch ein alternatives Design pro Kategorie. Es gibt zwei Blöcke pro Kategorie, insgesamt also sechs Blöcke und aktive RFID-Tags. Der Benutzer interagiert direkt mit diesen und platziert sie auf drei RFID-Lesegeräten.

Physisches User Interface: RFID Read & Launch System

Dieses System umfasst mehrere Komponenten, die alle an einen zentralen Raspberry Pi angeschlossen sind.

- RFID Reader:
  Sie sind verantwortlich für das Einlesen der RFID-Tags der Applikationsbausteine. Sobald sie ein Signal registrieren, senden sie über die Pins des Raspberry Pi ein Signal, das von diesem verarbeitet und als HTTPS-Anfrage an das Java-Backend gesendet wird.
- Deploy Knopf:
  Dieser Knopf ist ein Auslöser. Drückt man ihn, wird ein Deployment Prozess simuliert. Er ist an den Raspberry Pi angeschlossen.
- Raspberry Pi:
  Der Raspberry Pi ist mit den anderen Hardwarekomponenten verbunden. Es erkennt eingehende RFID Signale oder den Druck des Deployment Knopfes und kann auf Grundlage dieser Aktionen verschiedene Endpunkt im Java Backend aufrufen.

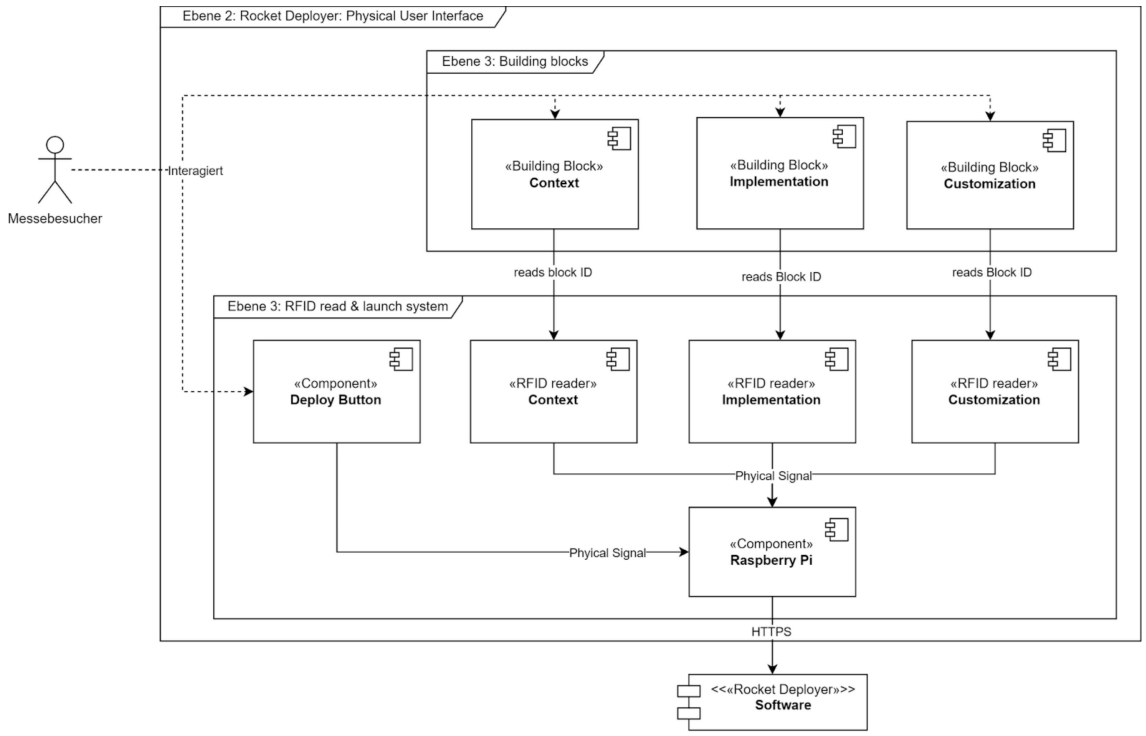


Abb. 6: Bausteinsicht Ebene 2: Physical User Interface

Java Spring Backend

Zum Zeitpunkt der Erstellung dieses Dokuments ist die genaue Struktur und Definition der Endpunkte der Applikation in der dritten Ebene noch nicht abschließend festgelegt.

Angular Frontend

Ähnlich wie beim Backend befindet sich das Frontend noch in der Ausarbeitungsphase. Allerdings ist bereits klar, dass eine Unterscheidung zwischen dem Hauptmonitor-UI und den individuellen Applikation-UI, die der Besucher durch die Kombination der Blöcke erstellt, erfolgen wird. Die genaue Realisierung dieser Unterscheidung ist noch nicht festgelegt.

Aufgrund der hohen Modularität des Frontends durch Angular wird in der Struktur der dritten Ebene nicht auf alle Komponenten einzeln eingegangen werden. Stattdessen wird lediglich die Unterteilung in Model, View und Controller vorgenommen und deren jeweilige Funktionsweise erläutert.

7. Laufzeitsicht

Die Laufzeitsicht wird durch drei Diagramme repräsentiert, welche exemplarisch die Hauptprozesse und Benutzerinteraktionen unserer Anwendung darstellen. Diese Prozesse spiegeln die Komponenten wider, die in der Ebene 1 und 2 der Bausteinsicht sichtbar sind, da die interne Struktur der Ebene 3 noch nicht ausreichend definiert ist.

7.1. Besucher legt Block auf Scanner

In diesem Diagramm wird der Prozess illustriert, bei dem ein Besucher einen Block auf einem der RFID-Leser platziert. Sobald er das tut, wird das Ereignis vom Raspberry Pi mittels eines HTTPS-Calls zum Backend weitergeleitet. Dieses teilt den aktuellen Zustand der Anwendung, also den zuletzt platzierten Blocks und die Liste der anderen gewählten Blöcke, mit allen registrierten Clients über einen Websocket-Kanal.

Das Hauptbenutzerinterface auf dem Monitor trifft dann die Entscheidung, welche Informationen dargestellt werden. In jedem Fall wird ein Informationsfeld mit Details über den ausgewählten Block angezeigt. Sollten insgesamt drei Blöcke auf den Lesegeräten platziert sein, wird zusätzlich ein Deploy-Indikator angezeigt. Dieser informiert den Besucher darüber, dass er nun die Anwendung abschließen kann.

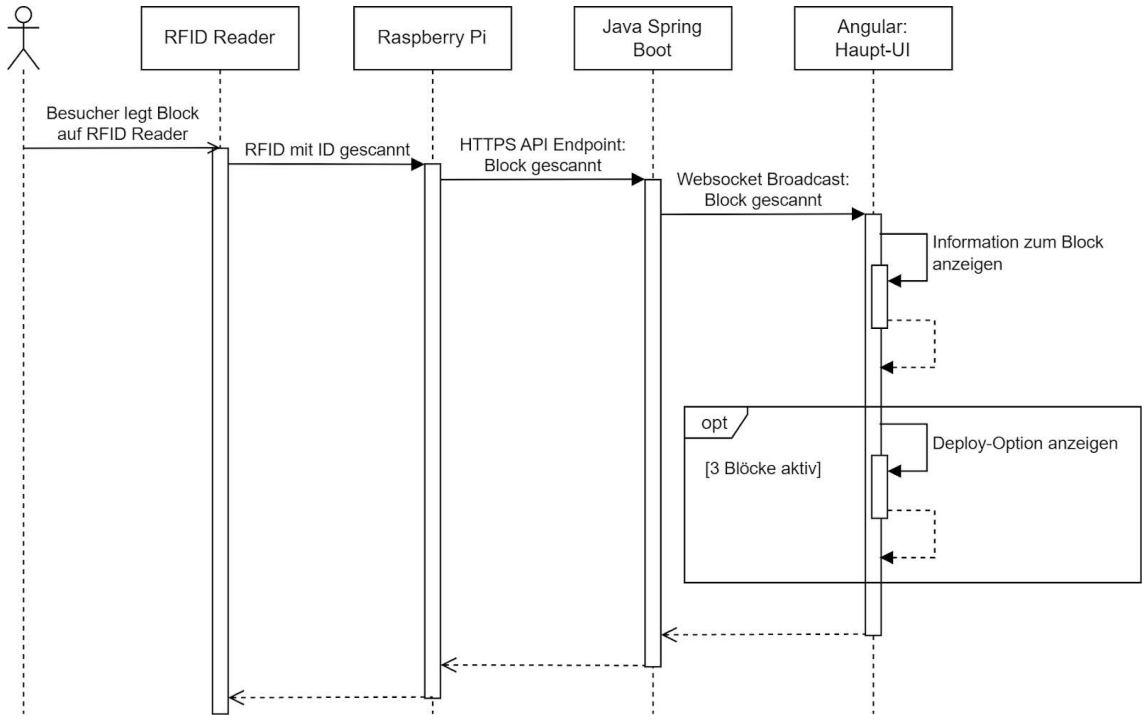


Abb. 7: Laufzeitsicht: Besucher legt Block auf Scanner

7.2. Besucher drückt Deployment Button

In diesem Ablaufdiagramm wird der Prozess dargestellt, wenn ein Besucher den Deploy-Button drückt. Der Raspberry Pi erkennt, dass der Button gedrückt wurde und sendet diese Information an das Java Backend. Das Backend prüft den aktuellen Zustand der Anwendung und leitet je nach Situation die passenden Schritte ein. Je nach Anzahl der aktiven Blöcke variiert die Reaktion des Systems:

- Wenn weniger als drei Blöcke aktiv sind, bleibt das System im aktuellen Zustand, da noch keine Applikation bereitgestellt werden kann.
- Sobald drei Blöcke aktiv sind und der Button zum ersten Mal gedrückt wird, wird der Haupt-Benutzeroberfläche mitgeteilt, dass der Konfigurationsbildschirm angezeigt werden muss. Auf dem Konfigurationsbildschirm wird dem Nutzer noch einmal die finale Konfiguration seiner Applikation gezeigt und ihm wird die Möglichkeit geboten, entweder gewählte Blöcke auszutauschen, oder mithilfe des Knopfs die Applikation zu deployen.
- Wenn der Besucher noch einmal den Knopf drückt, wird im Backend eine neue App-Konfiguration erstellt. Dies wird durch eine Insert-Anfrage an die PostgreSQL realisiert, bei der die ID des Eintrags zurückgegeben wird.
  Diese Information wird über einen Websocket-Broadcast an das Haupt-UI weitergegeben, das nun eine Launch-Animation abspielt und einen QR-Code basierend auf der übergebenen ID erzeugt und anlegt.

Die Bedingung “3 Blöcke sind aktiv” bezieht auch eine Überprüfung mit ein, dass es sich um eine valide Kombination an Applikations-Blöcken handelt und nicht etwa um zwei derselben Kategorie. Wenn dies der Fall wäre, würde ebenfalls nichts passieren.

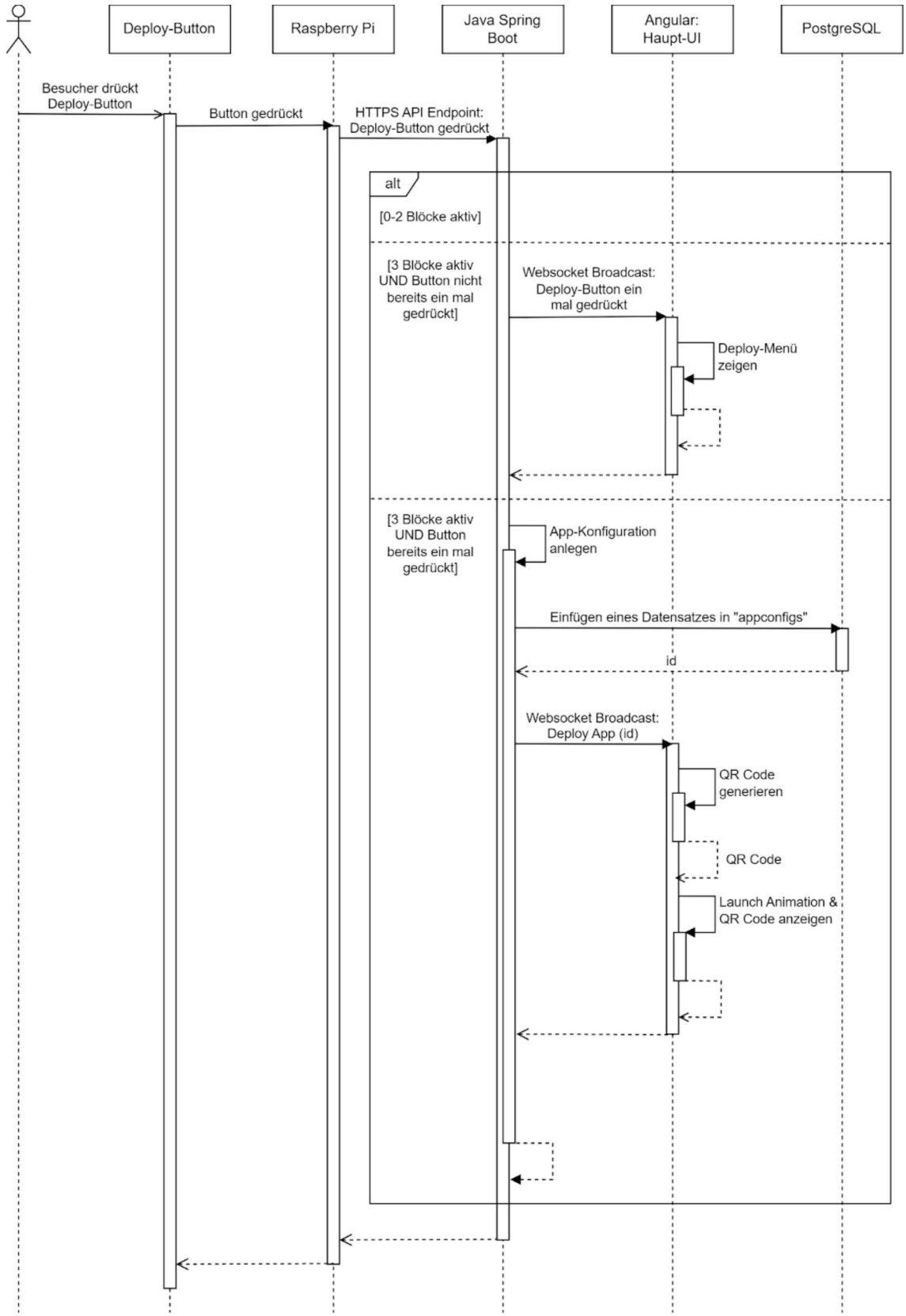


Abb. 8: Laufzeitsicht: Besucher drückt Deployment Button

7.3. Besucher ruft eine App-Konfiguration auf

Dieses Laufzeitdiagramm kann noch nicht angefertigt werden, da die genaue Implementierung der App-UIs und der verschiedenen Routen noch nicht feststeht. Wir sind derzeit dabei, die notwendigen Anforderungen und Funktionen zu ermitteln, um dieses wichtige Feature unserer Anwendung effizient zu gestalten. Sobald wir diese Details geklärt haben, können wir ein entsprechendes Laufzeitdiagramm erstellen.

8. Verteilungssicht

Die Verteilungssicht beschreibt, wie die verschiedenen Komponenten des Applikations-Baukastens auf verschiedene Hardware-Ressourcen verteilt sind und wie sie miteinander kommunizieren. Die darin sichtbaren Komponenten referenzieren die Komponenten aus der Bausteinsicht.

- **Cloud Foundry**: Die eigens geschriebenen Cloud-basierten Komponenten des Systems, also das Java-Backend und AngularJS-Frontend, werden in einer Cloud Foundry Umgebung auf der SAP BTP gehostet. Die SAP BTP stellt sicher, dass die Kommunikation zwischen den Komponenten effizient und sicher ist.

  Das Java-Backend und das Angular-Frontend interagieren miteinander über RESTful API und Websockets. Das Java-Backend stellt verschiedene Endpunkte zur Verfügung, die vom Frontend und vom Raspberry Pi aufgerufen werden. Gleichzeitig kommuniziert das Backend mit der PostgreSQL-Datenbank und den anderen Services auf der SAP BTP, um beispielsweise den Zustand der Anwendung und die Nutzerdaten zu verwalten.

- **Messestand**: Die physischen Komponenten des Systems, also der Raspberry Pi, RFID Reader, RFID-Tags und der Deploy Button, sind direkt miteinander verbunden. Der Raspberry Pi kann die Zustände der RFID-Tags einlesen. Die Python-Software auf dem Raspberry Pi kommuniziert mittels Web-Requests mit dem Java Backend, um den Zustand der RFID-Tags und den Status des Deploy-Buttons zu übermitteln.

- **SAP BTP Services**: Die SAP BTP stellt die zentralen Backend- und Datenbankdienste für das System bereit. Diese Services sind die PostgreSQL-Datenbank, die SAP Conversational AI, der xsuaa (Authorization Service) und ein bisher nicht genauer definierter AI Service, auf den eine der Applikationen aufbauen wird.

- **Messebesucher Mobiles Endgerät**: Das Endgerät des Benutzers ist ein Smartphone oder ein anderes Gerät mit einem Webbrowser, das zur Anzeige der von den Messebesuchern erstellten Web-Applikationen verwendet wird. Diese Geräte kommunizieren über das Internet mit dem Angular Frontend auf der SAP BTP, um die erstellten Applikationen zu laden und anzuzeigen. Die darauf folgende Kommunikation erfolgt über HTTPS und Websockets.

Diese Verteilung ermöglicht es den Benutzern, eine direkte und interaktive Erfahrung mit dem System zu haben, da die Applikation nicht nur auf einem Gerät, sondern auf einem Monitor, mit haptischen Baublöcken und dem Endgerät des Besuchers stattfindet.

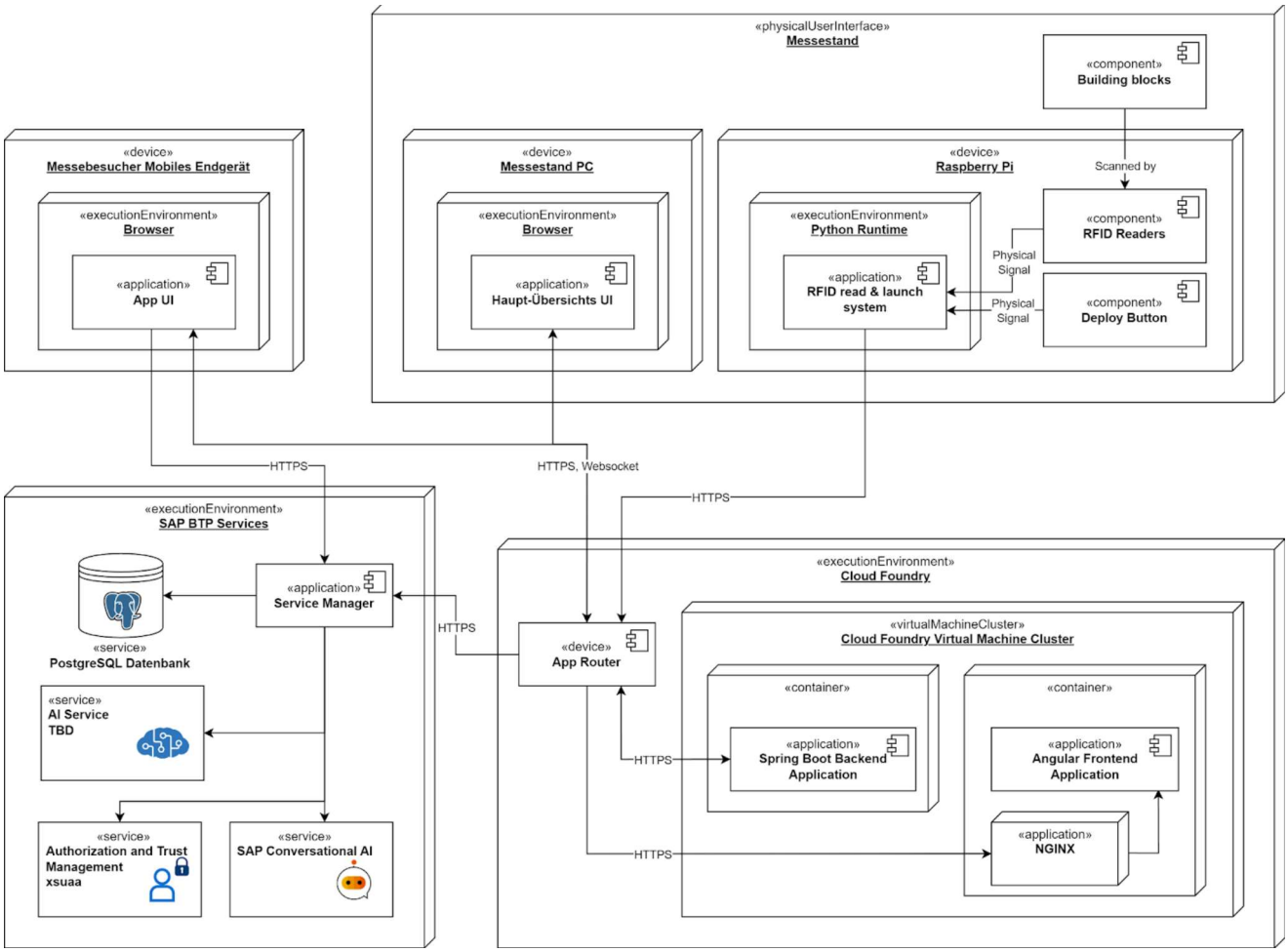


Abb. 9: Verteilungssicht

9. Architekturentscheidungen

Architekturentscheidungen sind in den Kapiteln Technologieentscheidungen und Architekturmuster bereits aufgeschlüsselt.

10. Qualitätsanforderungen

Die folgenden Qualitätsanforderungen können in Kapitel 8 der Anforderungsspezifikation (Pflichtenheft), welche am 23.05.23 um 17:00Uhr zur Verfügung gestellt wird, gefunden werden: https://team-leap.de/documents/Pflichtenheft_v2.0.pdf.

Es werden lediglich Titel und Beschreibung der Anforderung gezeigt. Priorität, Fit-Kriterien und weitere Details sind in der Anforderungsspezifikation zu finden.

10.1 Fehlertoleranz Software

Das System ist in der Lage, Softwarefehler zu erkennen, zu isolieren und zu behandeln, um ein vollständiges Systemversagen zu vermeiden.

10.2 Skalierbarkeit der Nutzer-Applikationen

Wenn ein Nutzer eine Applikation erstellt hat, soll diese von mehreren Nutzern gleichzeitig verwendet werden können, ohne dass diese einander beeinflussen.

10.3 Uptime

Die "App Builder" Applikation, sowie die von den Nutzern erstellten Applikationen müssen während der gesamten Dauer einer Messe verfügbar sein. Bei Abstürzen soll die Applikation innerhalb von 5 Minuten neu gestartet werden können.