Architekturdokumentation nach Arc42

Semesterprojekt im SS 2023

in Kooperation mit sovanta AG

Produkt “Rocket Deployer”

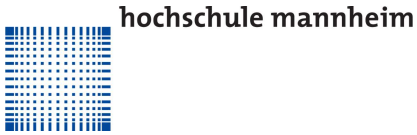
Team leap

Version 2.0

20.06.2023

Verantwortliche

Yan Wittmann, Lauritz Fuchs



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
v2.1	20.06.2023	- kleine Änderungen	Yan Wittmann Lauritz Fuchs
v2.0	19.06.2023	- Umsetzen des Dozenten-Feedbacks - Klassendiagramme erstellt - Laufzeitdiagramme überarbeitet - Änderungen an der Architektur seit letzter Version im Dokument anpassen	Yan Wittmann Lauritz Fuchs
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

1. Einführung und Ziele

1.1. Einleitung

Der Auftragnehmer will im Rahmen des Semesterprojektes 2023 an der Hochschule Mannheim eine Applikation erstellen, welche sowohl Neukunden, als auch SAP Business Technology Platform (BTP) und SAP (On-premise) Bestandskunden, davon überzeugen soll, mithilfe von Dienstleistungen der sovanta AG in die SAP BTP zu wechseln bzw. diese effizienter zu Nutzen.

Die Applikation soll potenziellen Kunden an einem Messestand der sovanta AG einen Einblick verschaffen, was mithilfe der SAP Business Technology Platform (BTP) möglich ist und dessen Funktionen auf spielerische und interaktive Art und Weise erklären.

Dieses Dokument dient der Beschreibung und Erklärung der während der Entwicklung unseres Produktes, Rocket Deployer, entstandenen Architektur. Zu diesem Zwecke werden in den folgenden Kapiteln auf verschiedenen Architekturebenen und Sichten, als auch Kontext, Rahmenbedingungen und weitere für die Architektur relevante Themenpunkte eingegangen.

1.2. Auftraggeber

Der Auftraggeber, sovanta AG, ist eine Aktiengesellschaft mit Sitz in Heidelberg, welche im IT-Bereich diverse Dienstleistungen für ihre Kunden anbietet. Die Haupttätigkeiten hierbei sind die Entwicklung von Business Software Solutions aufbauend auf Produkten der SAP, um die Bedienung als auch Darstellung der SAP Business Suite für Endnutzer zu vereinfachen.

1.3. Aufgabenstellung

Die sovanta AG, unser 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.

1.4. Produktvision

Im Rahmen des Projekts möchten wir einen interaktiven Baukasten für Webapplikationen entwickeln, den sog. Rocket Deployer. Dieser soll Anwender in die Lage versetzen, physische Disketten, welche einzelne Services der SAP BTP oder die Funktionalität einer Applikation repräsentieren (bspw. ToDo-App), zu kombinieren und zu einer lauffähigen Webanwendung zusammenzusetzen. Jede Diskette ist hierbei mit einem RFID-Tag versehen, welche mithilfe von RFID-Readern eingescannt und identifiziert werden können.

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.

1.5. 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.

1.6. Stakeholder

1.6.1. 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	• Tutor Ivo Seitz	• 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

1.6.2. 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

2. 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.

2.1. Technische Randbedingungen

RB1: SAP BTP als Zielplattform

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

RB2: Einschränkung der erlaubten Open-Source Lizenzen

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

RB6: Niedrige Komplexität des Applikations-Codes

Der Kunde will, dass keine technisch beeindruckende Lösung gebaut wird, da Messebesucher vom Backend nichts mitbekommen werden und eher Wert auf das Erlebnis des Besuchers gelegt werden soll. Daher sollen Teile der Applikation, die normalerweise komplex zu implementieren wären, mit nachgebildeten und simulierten Komponenten ersetzt werden.

2.2. Finanzielle Randbedingungen

RB3: Eingeschränkte Ressourcenbereitstellung durch den Kunden

Das Projekt ist insofern besonders, dass der Kunde kein pauschales Projektbudget (in Euro) zur Verfügung stellt. 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

2.3. Organisatorische Randbedingungen

RB4: 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.

RB5: Vorgegebenes Vorgehensmodell - Eingeschränkte Planungsfreiheit

Die Nutzung von Scrum, zumindest die Durchführung von Sprints, wird durch die Professoren stark suggeriert. Wenn sich für Scrum entschieden wird, 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.

3. Kontextabgrenzung

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

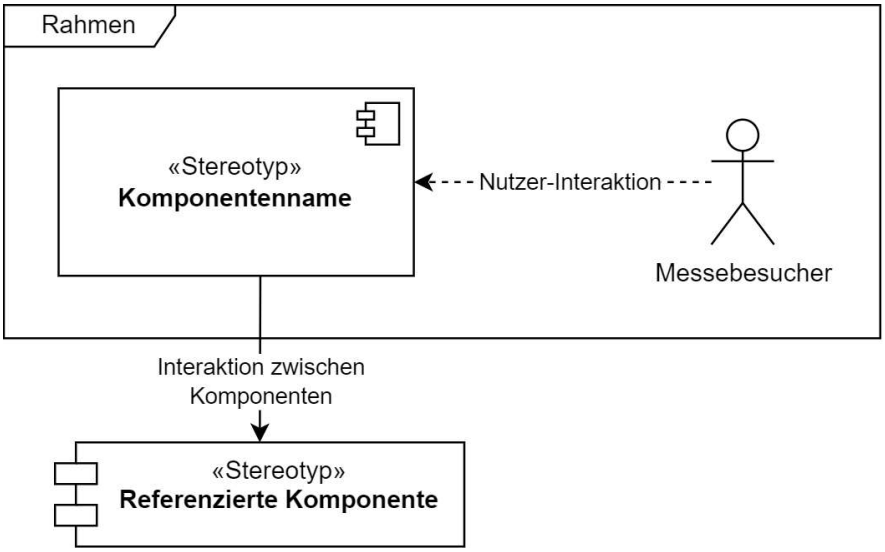


Abb. 12: Erläuterung der Diagrammelemente

3.1. Fachlicher Kontext

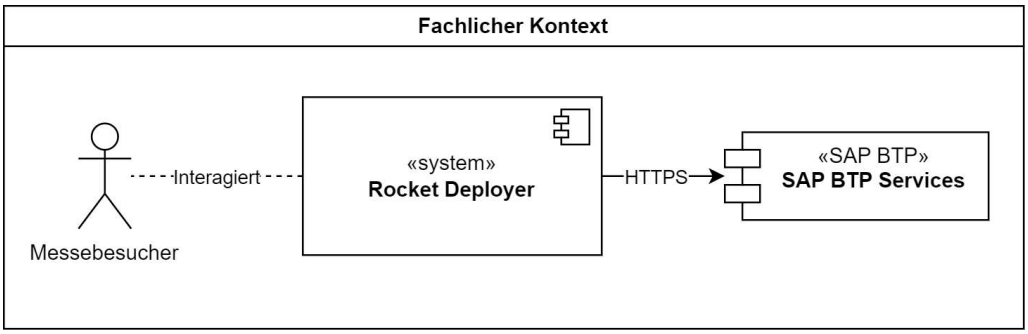


Abb. 1: Fachlicher Kontext

3.1.1. Rocket Deployer

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

3.1.2. 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 Messebesucher eine Anwendung zusammengestellt hat und diese erstellt wurde, kann er sie auf seinem mobilen Endgerät aufrufen und benutzen.

3.1.3. 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.

3.2. Technischer Kontext

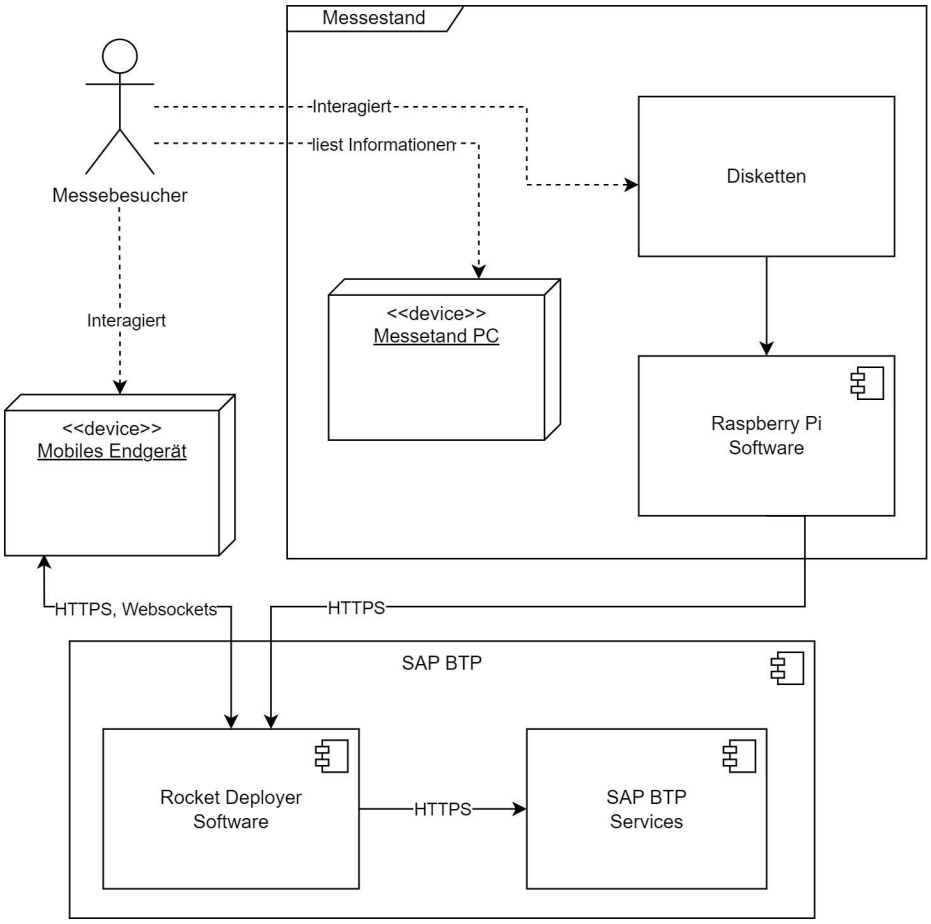


Abb. 2: Technischer Kontext

3.2.1. Messestand PC

Der Messestand PC dient zur Anzeige von Informationen über die letzte Diskette, 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. Nach dem Bauprozess wird ein QR-Code angezeigt, um diese Applikation nutzbar zu machen.

3.2.2. Raspberry Pi

Der Raspberry Pi scannt mit RFID Readern die Disketten 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.

3.2.3. Disketten

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

3.2.4. SAP BTP

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

3.2.5. Mobiles Endgerät

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

4. Lösungsstrategie

4.1. Qualitätssicherung

4.1.1. 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 die SAP BTP hochgeladen, sodass sie aus dem Internet erreichbar ist.
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.

4.1.2. Kontinuierliche Integration und Deployment (CI/CD)

Unsere CI/CD-Pipeline wird durch GitLab (https://docs.gitlab.com/ee/ci/) 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 Unit-Tests im Java Backend und die automatisch generierten Spezifikationstests im Angular Frontend 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) (https://aws.amazon.com/de/ec2/) einen Custom GitLab Runner installiert und konfiguriert, der unsere Pipelines ausführen kann.

4.1.3. 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.

4.1.4. Teststrategie

4.1.4.1. Angular Frontend

Im Angular Frontend setzen wir das Framework Jest (https://jestjs.io/) ein, ein Behavior-Driven Development (BDD) Framework für JavaScript. Jest wird verwendet, um verschiedene Aspekte der Frontend-Logik zu testen. Allerdings werden hier nur automatisch generierte Spezifikations-Tests ausgeführt, da aufgrund von Zeitmangel und anderen Prioritäten hier im Frontend eher auf Nutzer-Tests gesetzt wird.

4.1.4.2. Java Backend

Im Rahmen unserer Teststrategie setzen wir verschiedene gängige Testwerkzeuge und Frameworks aus dem Java-Ökosystem ein, darunter auch JUnit 5 (https://junit.org/junit5/). Dieses Framework bietet uns eine Möglichkeit, Modultests zu erstellen und einzelne Komponenten unserer Anwendung zu überprüfen.

Unser Ziel ist es, eine Zweigabdeckung von 70% zu erreichen. Diese Quote wurde ausgewählt, da wir uns bewusst sind, dass eine hundertprozentige Abdeckung in der Praxis oft nicht realistisch ist. Insbesondere einfache Getter- und Setter-Methoden, Initialisierungscode, Produktionsdatenbank-Adapter und WebSocket-Handler sind Beispiele für Codeblöcke, die aufgrund ihres begrenzten Mehrwerts üblicherweise vom Testen ausgeschlossen werden.

Angesichts unserer aktuellen Projektgeschwindigkeit und der damit verbundenen Implementierungs- und Dokumentationsaufwände schätzt unser Team eine höhere Abdeckungsquote als nicht realistisch ein. Wir möchten vermeiden, dass durch eine zu ambitionierte Zielsetzung wichtige Aspekte vernachlässigt werden. Unsere bisherige

Testabdeckung gibt uns jedoch Zuversicht, dass die angestrebte Quote von 70% durchaus erreichbar ist.

5. Bausteinsicht

5.1. Ebene 1

Der Rocket Deployer ist eine Lösung, die sowohl physische als auch softwarebasierte Komponenten umfasst. Die oberste Ebene der Bausteinsicht des Rocket Deployer Systems (Abb. 3) ist in zwei Hauptkomponenten unterteilt: die physische Benutzerschnittstelle und das Backend/Frontend.

Der Anwender interagiert mit beiden System:

- Er nimmt die physischen Disketten 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 (Abb. 5) weiter aufgeschlüsselt.

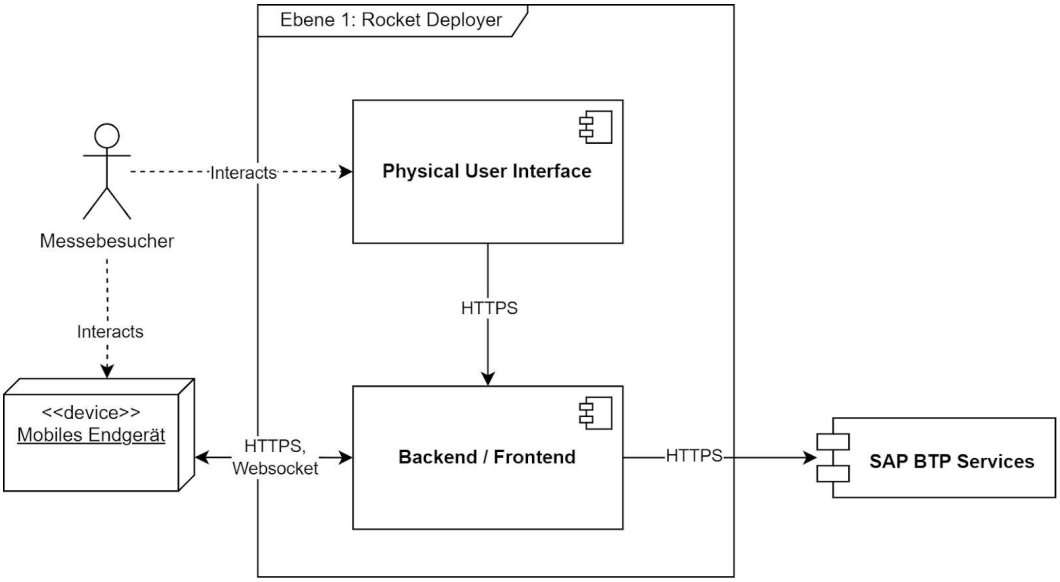


Abb. 3: Bausteinsicht Ebene 1

5.2. Ebene 2

Im Abb. 4 werden sowohl das Physical User Interface, als auch das Frontend/Backend auf der SAP BTP aufgeschlüsselt.

Als Kommunikationsprotokolle werden zwischen den einzelnen Komponenten HTTPS und WebSocket Kanäle verwendet. HTTPS wird für die Kommunikation zum Java Backend und Angular Frontend, das auf der SAP BTP gehostet wird, verwendet und WebSockets, um eine bidirektionale Kommunikation zwischen dem Mobilen Endgerät und dem Backend zu ermöglichen.

Die SAP BTP verwendet einen App Router, um über die Domain und den Host zu entscheiden, an welche Applikation (Java Backend, Angular Frontend) die Requests geleitet werden sollen.

Physical User Interface

Die physische Benutzerschnittstelle, mit der der Messebesucher interagiert, besteht aus den Applikations-Disketten, die der Besucher auf die RFID-Reader legt. Jede dieser Disketten enthält einen RFID-Tag, der sich mittels eines eindeutigen Identifiers gegenüber dem RFID-Reader identifiziert, wenn die Diskette auf den Reader gelegt wird.

Zusätzlich besteht die physische Benutzerschnittstelle aus einem Deploy-Knopf und einem Schlüssel, deren Zustände ausgelesen werden können. Nachdem der Besucher seine Auswahl an Disketten getroffen hat, kann er den Schlüssel drehen, um eine Konfiguration zu “locken” und danach den Deploy-Knopf drücken, um diese zu starten. Diese Events werden ebenfalls vom Raspberry Pi erfasst und an das Java-Backend gesendet.

SAP BTP: Backend / Frontend

Die Softwarekomponente des Systems, die auf der SAP BTP läuft, besteht aus mehreren Subkomponenten.

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 mit dem PostgreSQL-Dienst auf der SAP BTP (Abb. 5), um die notwendigen Daten zu speichern und zu verwalten. Es bietet auch Endpunkte für das Angular Frontend:

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.

Ein anderer Teil des Angular-Frontends ist 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-Dienst zuzugreifen, der auf der SAP BTP gehostet wird.

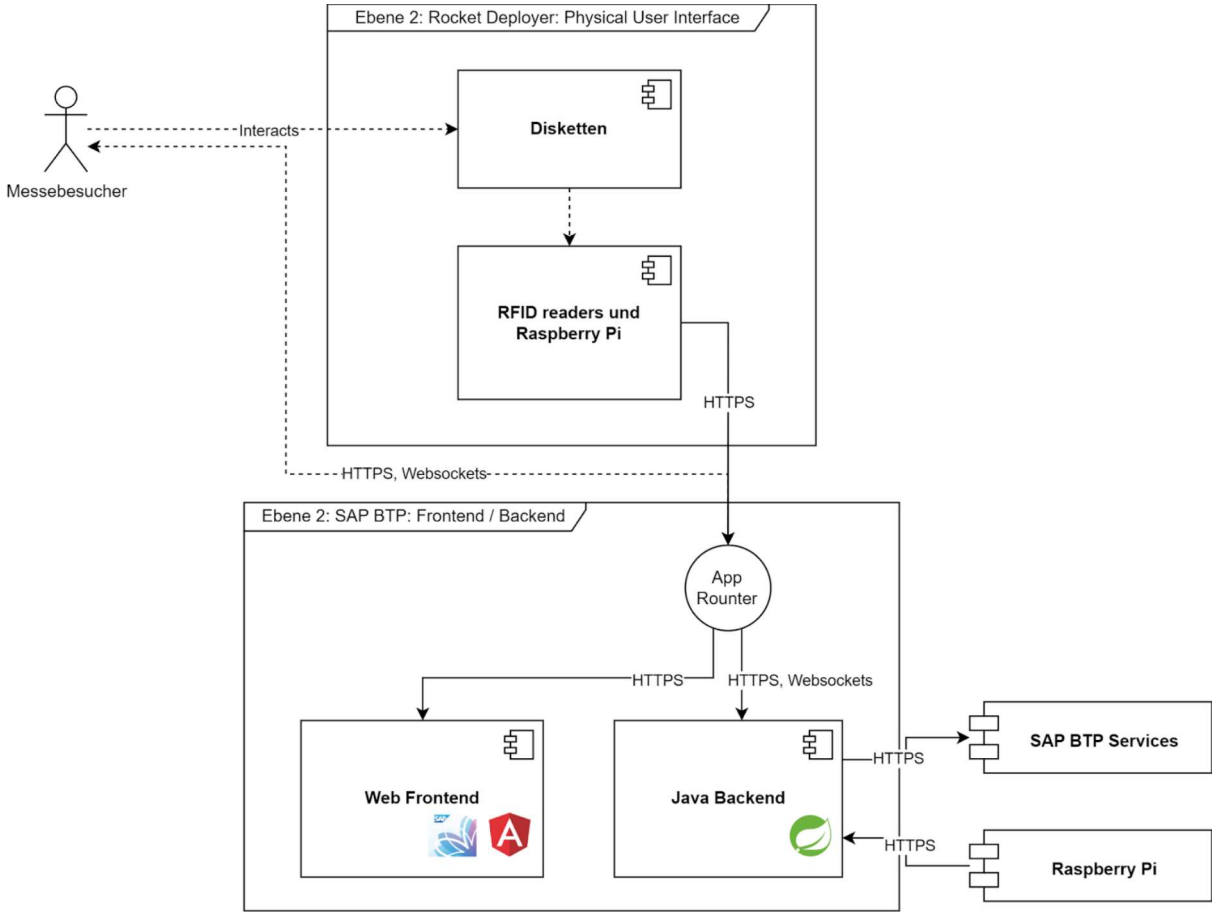


Abb. 4: Bausteinsicht Ebene 2: Rocket Deployer

SAP BTP: Services

In Abb. 5 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.

Im Gegensatz zu Version v1.0 der Architekturdokumentation wird in der aktuellen Implementierung nur noch ein Service verwendet. Dies ist aus Kundengesprächen hervorgegangen: Die Lösung des Problems sollte technisch möglichst einfach sein, daher sollen Services wie ein Auth Service durch eine einfache Benutzerverwaltung im Backend ersetzt werden.

Damit verbleibt noch der PostgreSQL Database-Service: Details zu PostgreSQL und warum wir es nutzen, werden im Kapitel „Technologieentscheidungen“ erläutert. Eine genaue Auflistung der Datenbanken, der Schemas und Erklärungen kann auf Ebene 3 gefunden werden.

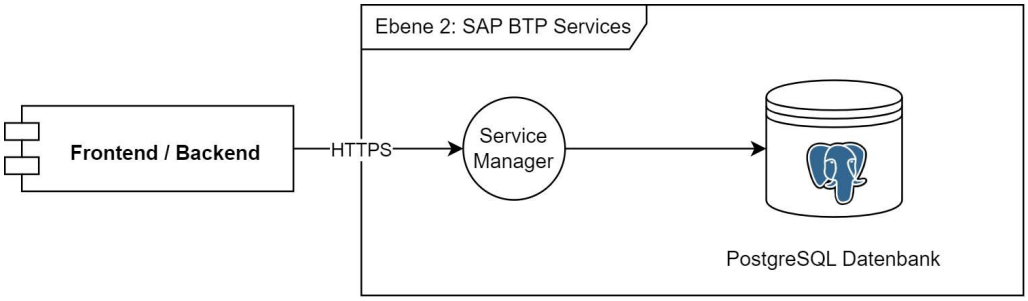


Abb. 5: Bausteinsicht Ebene 2: SAP BTP Services

5.3. Ebene 3

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

Physisches User Interface: Disketten

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 Disketten sind visuell unterscheidbar durch ein alternatives Design pro Kategorie. Es gibt zwei Disketten pro Kategorie, insgesamt also sechs Disketten mit RFID-Tags. Der Benutzer interagiert direkt mit diesen und platziert sie auf drei RFID-Lesegeräten.

Physisches User Interface: RFID readers und Raspberry Pi

Dieses System umfasst mehrere Komponenten:

- 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 Schlüssel:
  Jedes Hin-/Herdrehen dieses Schlüssels (jede Flanke) wird dem Backend mitgeteilt und löst das “Locken” der App-Konfiguration aus. Er ist an den Raspberry Pi angeschlossen.
- Deploy Knopf:
  Dieser Knopf ist ein Auslöser. Drückt man ihn, wird die ge-”Lockte” App Konfiguration als QR-Code verfügbar gemacht. 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.

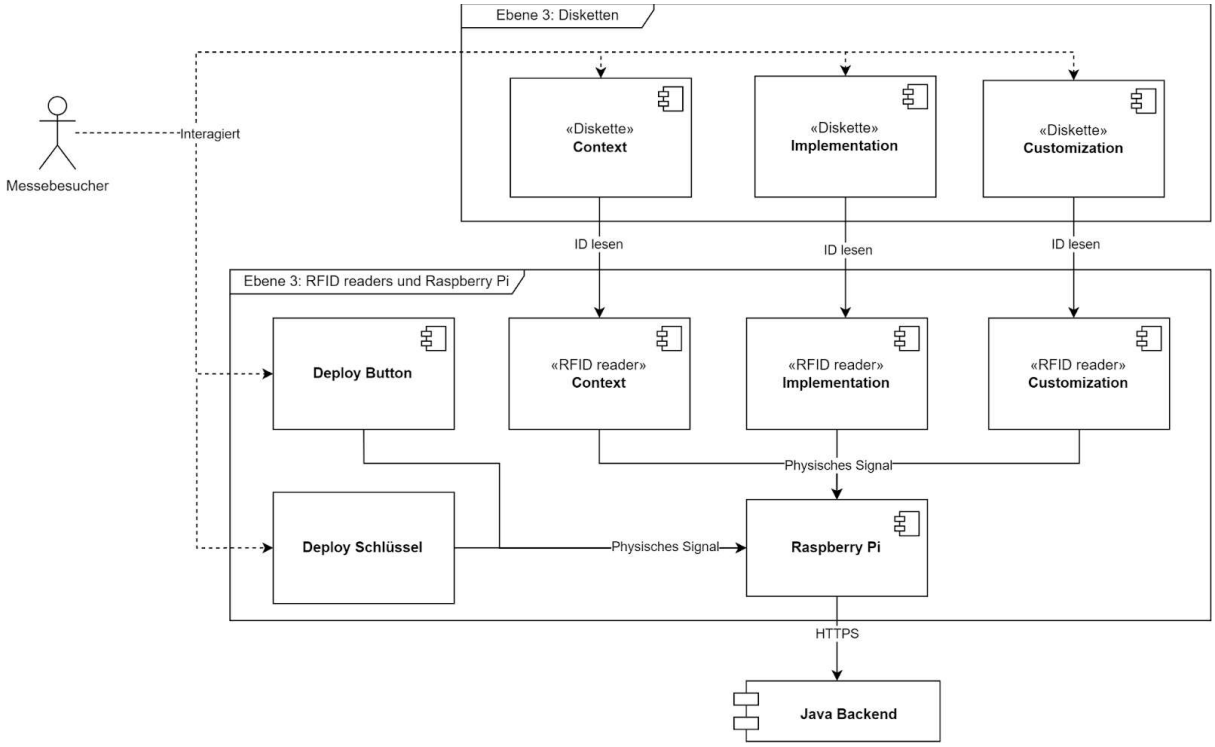


Abb. 6: Bausteinsicht Ebene 3: Physical User Interface

PostgreSQL Datenbank-Service

Dieser Service wird von der SAP BTP bereitgestellt. Indem man ihn in der manifest.yml-Datei das Java Backend als “Bound Application” registriert, erhält das Backend ab sofort bei jedem Applikationsstart eine Umgebungsvariable VCAP_SERVICES mitgegeben, die Zugangsdaten enthält, mit denen auf die Datenbank zugegriffen werden kann, sofern die Applikation im lokalen Netzwerk auf der BTP läuft, also mit Cloud Foundry deployed wurde.

Unsere Schemata, die wir auf der Datenbank verwenden, können im resources/db/schema-Verzeichnis gefunden werden. Im Folgenden und Abb. 15 sind die Tabellen aufgeschlüsselt.

- appconfig: Speichert die vom Messebesucher erzeugten App-Konfigurationen.
- kanban: Repräsentiert ein Kanban-Board mit Aufgaben, die in Kategorien eingeteilt und priorisiert werden können.
    - Der "status"-Wert muss zwischen 0 und 2 liegen.
    - Der "priority"-Wert muss zwischen 0 und 2 liegen.
- learned_blocks: Speichert eingelernte Blöcke.
- todo: Repräsentiert eine Aufgabenliste mit Einträgen, die erledigt werden können.
- users: Speichert die Benutzer, die sich mit der Loginmaske registriert haben.

- users_tokens: Speichert Benutzertokens, die von angemeldeten Nutzern erzeugt wurden.

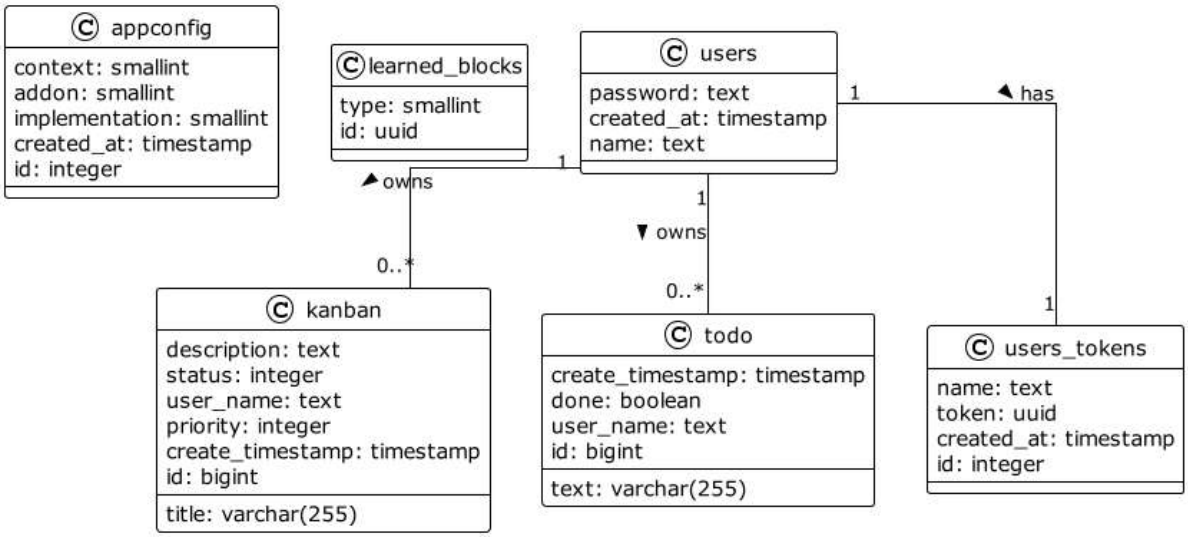


Abb. 15: PostgreSQL-Datenbankschema

Java Spring Backend

Im Folgenden wird auf den Aufbau des Java Backends eingegangen. Die Erklärung erfolgt in den Abschnitten:

- Datenbank-Adapter
- DAO-Pattern Klassen
- Services
- REST-API Controller
- Application state

Abbildung 10 zeigt das Klassendiagramm in einer Übersicht.

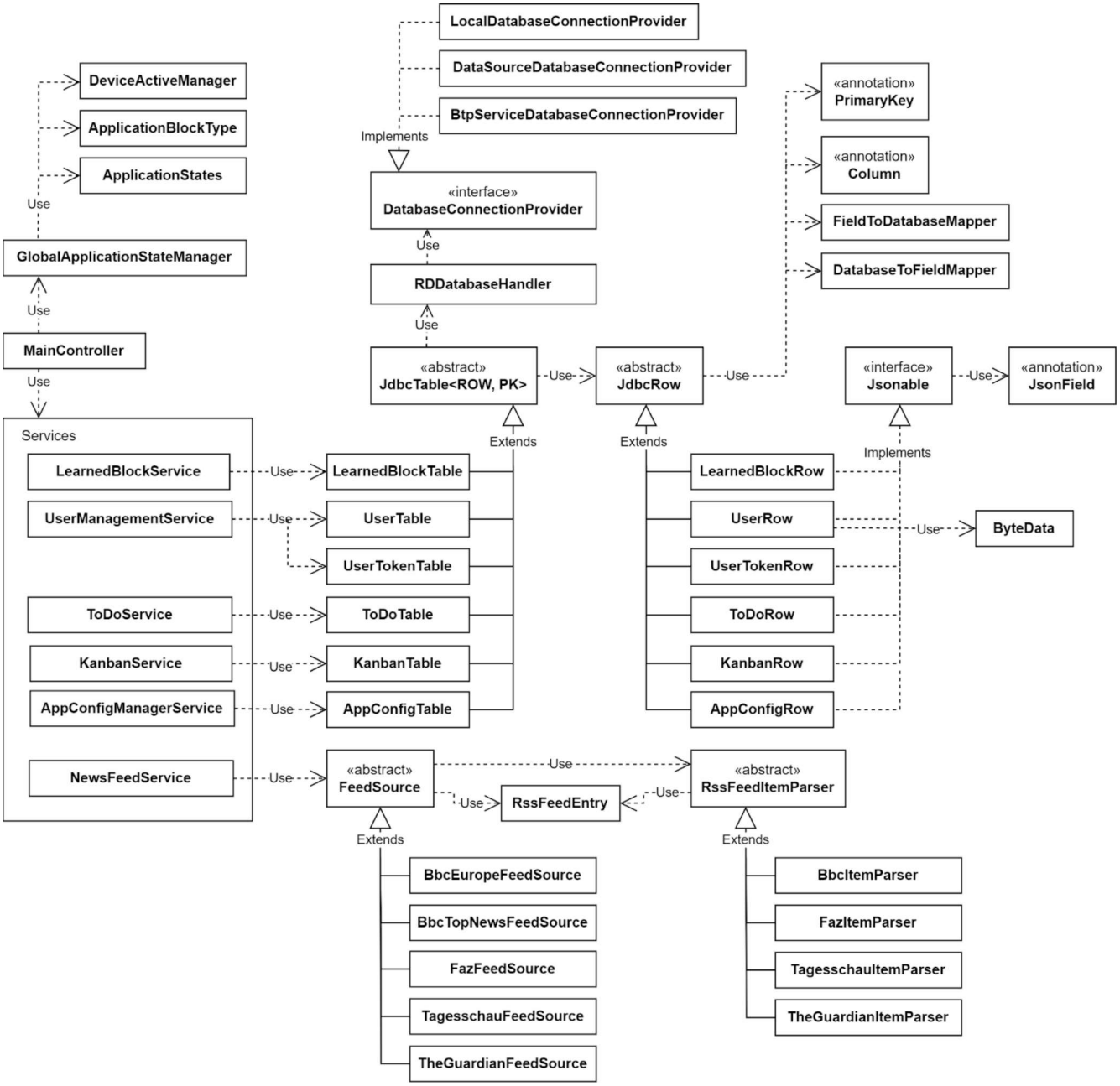


Abb. 10: Klassendiagramm Java Backend

Datenbank-Adapter

Die Klasse RDDatabaseHandler stellt ein Interface dar, um je nach Umgebung, in der die Applikation ausgeführt wird, DatabaseConnectionProvider-Instanzen zu erzeugen, zurückzugeben und zu verwalten. Dies ist notwendig, da je nach Umgebung (Lokal, Test, SAP BTP) eine unterschiedliche Datenbank zum Einsatz kommt und dies an einem Ort behandelt werden soll. Die Entscheidung, welche Datenbank zum Einsatz kommt, wird folgendermaßen getroffen:

1. Lokale Datenbank:
   Wenn die Property app.database.local.uri in der Maven POM gesetzt ist.
2. SAP BTP:
   Wenn es die Umgebungsvariable VCAP_SERVICES gibt und eine app.database.service.id in der Maven POM gesetzt ist.
3. Test-Datenbank:
   Wenn EmbeddedPostgresDbProvider.registerEmbeddedDb() in einer Testklasse aufgerufen wird.

DAO-Pattern Klassen

Wie in den Architekturentscheidungen begründet, verwenden wir nicht die Implementierung des DAO Patterns von Spring, sondern eine eigene Implementierung. Hierfür gibt es zwei entscheidende Basisklassen: JdbcTable und JdbcRow.

Implementierungen von JdbcTable sind für das verwalten einer Tabelle zuständig: Sie geben eine Referenz zum Schema im resources-Ordner, den Tabellennamen, den Primärschlüssel und dessen Typs und können Instanzen der entsprechenden JdbcRow-Implementierung aus ResultSet s von JDBC-Queries erzeugen.

Die Basisklasse JdbcTable verwendet diese Methoden und Informationen dann, um Abfragen auf der Tabelle in der Datenbank durchzuführen. Vor jeder Abfrage wird das Schema erzeugt, falls es nicht schon existiert. Es stellt einige Methoden zur Verfügung, wie alle Zeilen aufzulisten oder nach dem Primärschlüssel zu suchen. Für eigene Abfragen gibt es die getByPreparedStatement-Methode, die mittels eines Callbacks es ermöglicht, beliebige Abfragen auf die Tabelle zu machen.

JdbcRow-Klassen enthalten Attribute, die mit der @Column und ein Attribut davon mit der @PrimaryKey Annotation versehen sind. Diese Annotationen enthalten Metainformationen, die für Abfragen auf die Datenbank verwendet werden. Wichtig ist hier noch, dass die Setter nicht die Werte direkt setzen dürfen, sondern über die changeField-Methode gehen müssen, da so bei einer UPDATE-Anweisung nur die Attribute aktualisiert werden, die auch geändert wurden. Die Klassen sind in der lage, verschiedene Statements selbständig zusammenzubauen, wie INSERT, UPDATE und DELETE, welche dann auf der Table-Klasse ausgeführt werden können.

Implementierungen dieser Klassen haben (zumeist) je ein Gegenstück, es gibt also immer eine zueinander gehörende Table- und Row-Klasse.

REST-API Controller

Der MainController ist eine @SpringBootApplication und ein @RestController, der für das Annehmen der API-Calls verantwortlich ist. Hier ist ein beispielhafter Endpunkt zu sehen:

```
@GetMapping("/action/button")
@CrossOrigin(origins = "*")
public String getActionButtonPressed() {
    return exceptionResponseHandler(() -> {
        stateManager.deployButtonPressed(appConfigManagerService);
        return successJson();
    }, "Failed to process button press");
}
```

Um Exception-Handling in bestimmten Methoden einheitlich zu machen gibt es den exceptionResponseHandler, der im Falle einer Exception ein entsprechendes JSON generiert und als Rückgabe liefert, ansonsten wird das zurückgegebene JSON verwendet. Die Antwort enthält in jedem Fall eine success-Flag, die entweder true oder false ist. Wenn eine RDRequestException geworfen wurde, wird deren Nachricht auch noch eine error-Nachricht mit der Fehlermeldung mitgegeben, anstatt einer default-Nachricht.

Für Endpunkte mit Variablen im Pfad, wie /action/block/insert/{id}, wird mit einem Paramter-Attribut @PathVariable String id die Variable aus dem Pfad extrahiert.

Manche andere Endpunkte dürfen nur autorisiert aufgerufen werden, hier wird mit einem @RequestHeader(name = HttpHeaders.AUTHORIZATION, required = false) String token ein Header Authorization extrahiert.

Der MainController hat zudem Referenzen zu allen verwendeten Services, die im nächsten Abschnitt beschrieben werden und eine Referenz zu einer GlobalApplicationStateManager-Instanz, die den Applikationsstatus verwaltet.

Services

Einzelne abgeschlossene Funktionalitäten werden in Service-Klassen gekapselt. Diese haben keine einheitliche Super-Klasse, hierbei handelt es sich nur um ein umgesetztes Pattern. Es gibt je eine Instanz dieser Klassen, wenn die Applikation läuft (dennoch kein Singleton-Pattern).

Im Folgenden wird auf die einzelnen Services in mehr oder weniger Detail eingegangen.

- AppConfigManagerService
  Dieser Service verwaltet Applikationen, die mit dem Rocket Deployer gebaut wurden. Er kontrolliert nur begrenzt, ob es sich um valide Konfigurationen handelt, da dies schon vom State Manager verwaltet wird.
- LearnedBlockService
  Dient als einfaches Interface, um angelernte Blöcke abzufragen und zu speichern.
- UserManagementService
  Hat Referenzen auf zwei Tabellen: zur Verwaltung der existierenden Nutzer und der

eingeloggten Nutzer mit ihren Tokens. Es erlaubt Nutzer anzulegen, diese zu validieren und Tokens für validierte Nutzer zu erzeugen und diese dann wieder zu validieren. Die Passwörter werden in diesem Fall als Byte-Array mit SHA-256 ge-hasht, aber nicht ge-saltet.

- ToDoService
  Enthält Methoden zur Erstellung und Verwaltung von ToDo-Einträgen.
- KanbanService
  Kanban- und ToDo-Einträge ähneln sich sehr, daher ist die Implementierung hier sehr ähnlich zum ToDoService.
- NewsFeedService
  Hierbei handelt es sich um einen RSS-Feed Reader, der auf verschiedene Quellen konfiguriert werden kann. Er ist bisher der einzige Service, der keine Datenbank verwendet, da er keine benötigt. Die Einträge werden zehn Minuten lang gecached, um die Netzwerklast zu reduzieren.
  Um die RSS-Feeds abzufragen, werden Implementierungen der FeedSource Klasse verwendet. Diese stellen je eine Datenquelle dar, die durch RssFeedItemParser einzelne normalisierte Einträge erzeugen können.
  Diese können dann über die Endpunkte abgefragt werden.

Application state

Die Verwaltung des Applikations-Statuses ist eine der wichtigsten Aufgaben des Backends, da das Frontend zu jeder Zeit neu geladen werden kann, und daher der Status immer im Backend bekannt sein muss. Dies wird von der GlobalApplicationStateManager-Klasse getan.

Die Zustände, in denen unsere Applikation sein kann, sind die Folgenden. In diesem Zustand befindet sich die Applikation, wenn …

- INITIALIZING
  … die Applikation gerade gestartet hat, so lange bis alle Services gestartet sind. Danach wechselt sich der Zustand automatisch in BUILDING_WAITING.
- BUILDING_WAITING
  … keine Diskette eingeschoben wurde und man nicht im Lernmodus ist. Wenn ein Block draufgelegt wird, wechselt er in BUILDING_RUNNING. Wenn der Lernmodus gestartet wird, wechselt er in LEARNING.
- LEARNING
  … man im Lernmodus ist. Die Variable learningBlockStage zählt hier, bei welcher Diskette man gerade ist. Wenn die Variable den Maximalwert erreicht hat (ApplicationBlockType.NON_ADMIN_TYPES.length), dann wechselt er wieder in den BUILDING_WAITING Zustand.
- BUILDING_RUNNING
  … mindestens eine Diskette eingeführt wurde und die Applikation noch nicht deployed, also nicht der Schlüssel gedreht wurde.
- APP_DEPLOYED
  … wenn der Schlüssel gedreht wurde und eine valide Konfiguration vorliegt. Die Variable successiveDeployButtonPresses ergibt sich dabei aus [Schlüssel gedreht] +

[Button gedrückt] und gibt dem Frontend damit mit, in welcher deployment-Stage man sich befindet. Wenn die Applikation deployed ist, also successiveDeployButtonPresses = 2 und alle Disketten entfernt sind, wird wieder in den BUILDING_WAITING Zustand gewechselt.

Der Status wird über diverse Endpunkt-Aufrufe indirekt verändert, beispielsweise indem ein Block eingeführt oder ein Button gedrückt wird.

Ein einziges Mal pro Backend-Launch kann ein Token über /monitor/register abgefragt werden. Das Haupt-UI Frontend speichert diesen Token und kann dann damit durch einen autorisierten Request den Status abfragen.

Ein weiterer Wert, den der GlobalApplicationStateManager verwaltet ist, ob der Raspberry Pi online ist. Dies wird mit der DeviceActiveManager ermöglicht, die einen Heartbeat alle 10 Sekunden erwartet, ansonsten meldet dieser, dass der Pi offline ist.

Angular Frontend

Unser Angular Frontend ist in zwei Hauptbereiche unterteilt: die App-Konfigurationen und die Benutzeroberfläche für den Messe-Monitor. Jede App-Konfiguration liefert hierbei eine eigenständige Anwendung, die auf dem Smartphone des Benutzers ausgeführt werden kann. Die Ansicht auf dem Messe-Monitor beinhaltet das Interface zur Interaktion mit dem Deployment-Board.

Beide Ansichten bestehen aus Services und Komponenten, die in die Kategorien Apps, Monitor und Root eingeteilt sind. Diese Einteilung zieht sich durch unsere gesamte Anwendung und ist auch in der Ordnerstruktur wiederzuerkennen. Die Apps-Kategorie enthält alle Elemente, die für die Darstellung der App-Konfigurationen benötigt werden. Die Monitor-Kategorie enthält die Elemente, die für die Darstellung der Benutzeroberfläche auf dem Messe-Monitor benötigt werden. Die Root-Kategorie enthält grundlegende Softwareelemente, die sowohl in der Apps- als auch in der Monitor-Kategorie verwendet werden.

Die Kategorisierung hilft dabei, zu bestimmen, welche Services von welchen Komponenten genutzt werden können bzw. sollen. Im Allgemeinen nutzen Komponenten nur Services aus ihrer eigenen Kategorie oder aus der Root-Kategorie. Darüber hinaus hat jede spezifische App-Konfiguration in der Apps-Kategorie ihren eigenen zugehörigen Service. Dies erleichtert die potenzielle Erweiterung der Anwendung in der Zukunft.

Eine detaillierte Übersicht aller Softwareelemente und ihrer Kategorien wird bereitgestellt, aber eine grafische Darstellung der Beziehungen zwischen den Komponenten und Services wird aus Gründen der Übersichtlichkeit weggelassen. Stattdessen wird die Funktionsweise und Interaktion der einzelnen Elemente im Folgenden erläutert.

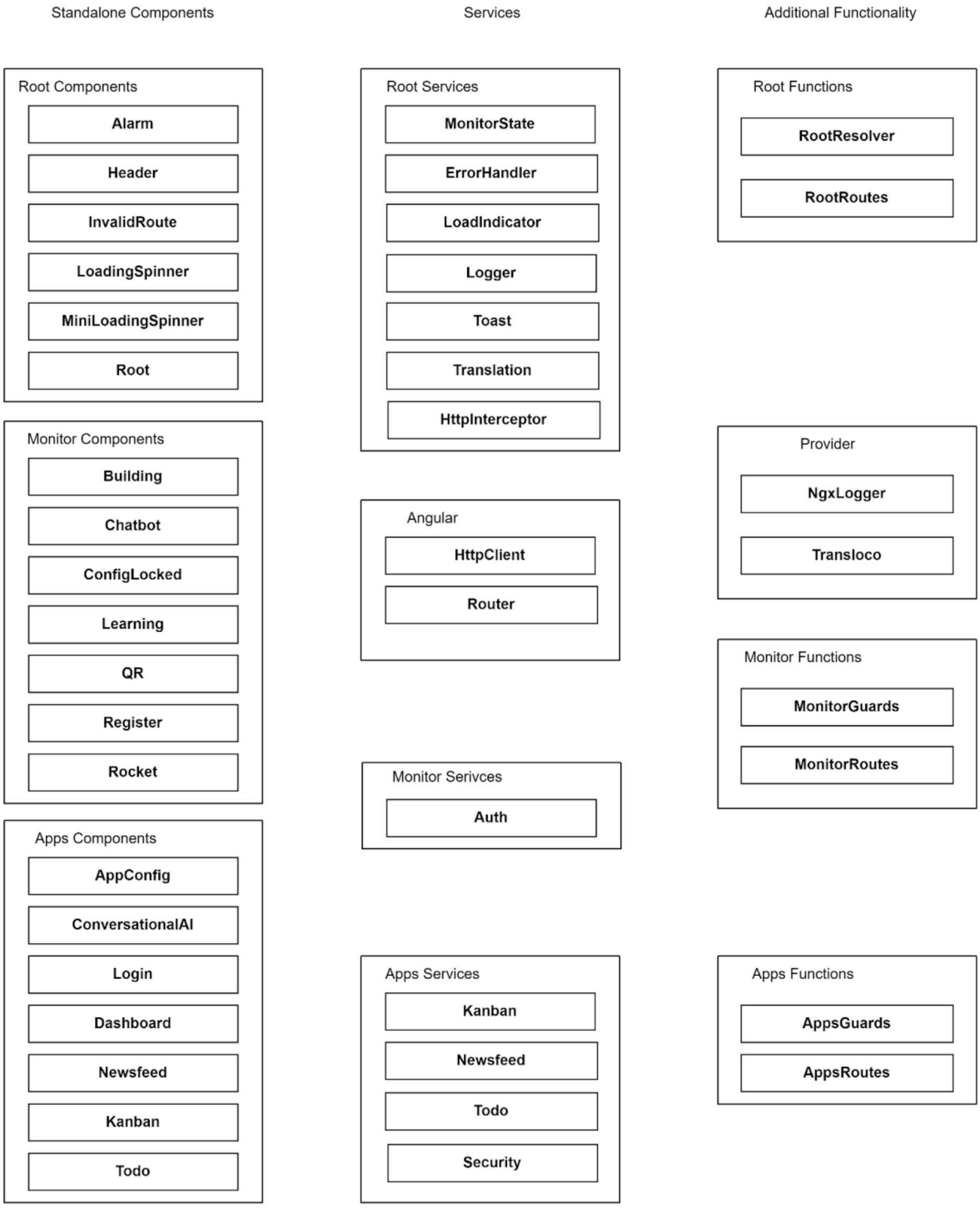


Abb. 11: Komponentenübersicht Angular Frontend

Zustandsmanagement

Für das Zustandsmanagement in unserem Frontend verwenden wir RXJS, eine Bibliothek, die Datenstrukturen und Funktionen bietet, um ein intuitives Zustandsmanagement im Angular Frontend zu ermöglichen.

Wir speichern alle Zustände und Interaktionen mit diesen in den Services, so auch die Umsetzung von HTTP-Anfragen an das Spring Boot Backend oder den Aufbau von WebSocket- Verbindungen. Alle Daten, die vom Backend kommen, werden in der RXJS-Datenstruktur BehaviorSubject gespeichert. Die BehaviorSubjects werden als Observables an die Komponenten übergeben, die damit auf die entsprechenden Daten zugreifen können. In den Komponenten erfolgt der Zugriff auf die Daten in den zugehörigen HTML-Dateien mithilfe von Async-Pipes. Die Async-Pipe ist ein Mechanismus von Angular, der automatisch ein Observable abonniert, wenn eine Komponente initialisiert wird, und das Abonnement automatisch beendet, wenn die Komponente zerstört wird.

Durch die Verwendung des Observable-Patterns und der Change Detection von Angular führen Änderungen im globalen Zustand automatisch zu einer Aktualisierung der Darstellung. Wir nutzen sowohl REST als auch WebSockets zur Kommunikation mit dem Backend. Das Request-Response- Modell wird bei allen Aktionen angewendet, die durch die Interaktion mit der Frontend-Benutzeroberfläche ausgelöst werden. Aktionen, die durch die Interaktion mit dem Deployment-Board ausgelöst werden, werden über WebSockets übermittelt. Wenn im Frontend WebSocket-Nachrichten empfangen werden, wird der lokale Anwendungsstatus entsprechend aktualisiert, um ihn synchron mit dem Anwendungsstatus im Backend zu halten.

Da Angular beim Neuladen einer Seite jeglichen temporären Speicher und somit auch den lokalen Anwendungsstatus löscht, wird nach jedem Neuladen der Status aus dem Backend angefordert und im Frontend wiederhergestellt. Da unser Frontend eine Single-Page-Anwendung ist, die bei Zustandsänderungen nicht vollständig neu geladen wird, beschränken sich die Backend-Anfragen zur Abfrage des Anwendungsstatus auf das manuelle Neuladen der Seite.

Routing

In der Monitor-Kategorie gibt es für jede Seite eine eigene Route. In der Apps-Kategorie sind die Routen wie folgt aufgebaut: "/app/appConfigId/configId/implementationId /contextId/customizationId". Hierbei befindet sich in der Route eine eindeutige appConfigId, welche eine gesamte Konfiguration identifiziert. Außerdem finden sich die Ids der verwendeten Bausteine wieder, aufgeteilt in deren jeweilige Kategorie.

Abhängig von der erstellten App-Konfiguration und den resultierenden Ids der einzelnen Bausteine wird beim Aufruf der Route innerhalb der AppConfig-Komponente, auf Grundlage der Ids, entschieden, welche Komponenten angezeigt werden.

Route-Guards

Die Route-Guards sind erforderlich, um die Authentifizierungsprozesse unserer Anwendung umzusetzen. Die Guards verhindern das Aufrufen von Routen, wenn der Client keinen

gültigen Token besitzt, der ihn zur Nutzung berechtigt. Dadurch gestatten wir nur sovanta-Mitarbeitern den Zugriff auf die Monitor-Benutzeroberfläche und können durch den Security Block Anwendungen erstellen, die eine eigene Benutzerverwaltung bieten.

6. Laufzeitsicht

Die Laufzeitsicht wird durch drei Diagramme repräsentiert, welche exemplarisch die Hauptprozesse und Benutzerinteraktionen unserer Anwendung darstellen.

Das Symbol in Abbildung 13 wird als Symbol für individuelle Tabellen in unserer PostgreSQL-Datenbank verwendet.

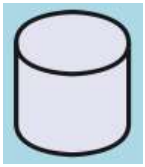


Abb. 13: Tabellen-Symbol

6.1. Besucher legt Diskette auf Scanner

In diesem Diagramm wird der Prozess illustriert, bei dem ein Besucher eine Diskette auf einem der RFID-Leser platziert. In diesem Szenario ist die Diskette die erste Diskette, die eingescannt wird, es ist eine bekannte und valide Diskette und keine besondere Karte, wie z.B. eine Admin-Diskette.

Sobald der Messebesucher seine Diskette einlegt, registriert der Raspberry Pi diese Änderung und sendet die einzigartige ID der Diskette an das Java Backend mittels eines HTTPS-Requests. Dieses leitet diese Aktion an den State Manager weiter, der nun entscheidet, was für eine Zustandsänderung dies bedeutet. Da es sich nicht um eine Admin-Karte handelt und der Zustand BUILDING_WAITING ist, wird er nun versuchen, den dazugehörigen angelernten Block aus der Datenbank zu finden. In diesem Szenario findet er einen und fügt ihn zu der Liste der aktiven Blöcke hinzu und wechselt den Zustand zu BUILDING_RUNNING. Nun bleibt nur noch übrig, dies dem Angular Frontend mitzuteilen: Über einen WebSocket Kanal sendet er eine entsprechende Botschaft, welche das Frontend verarbeitet und auf dem Building-UI darstellt.

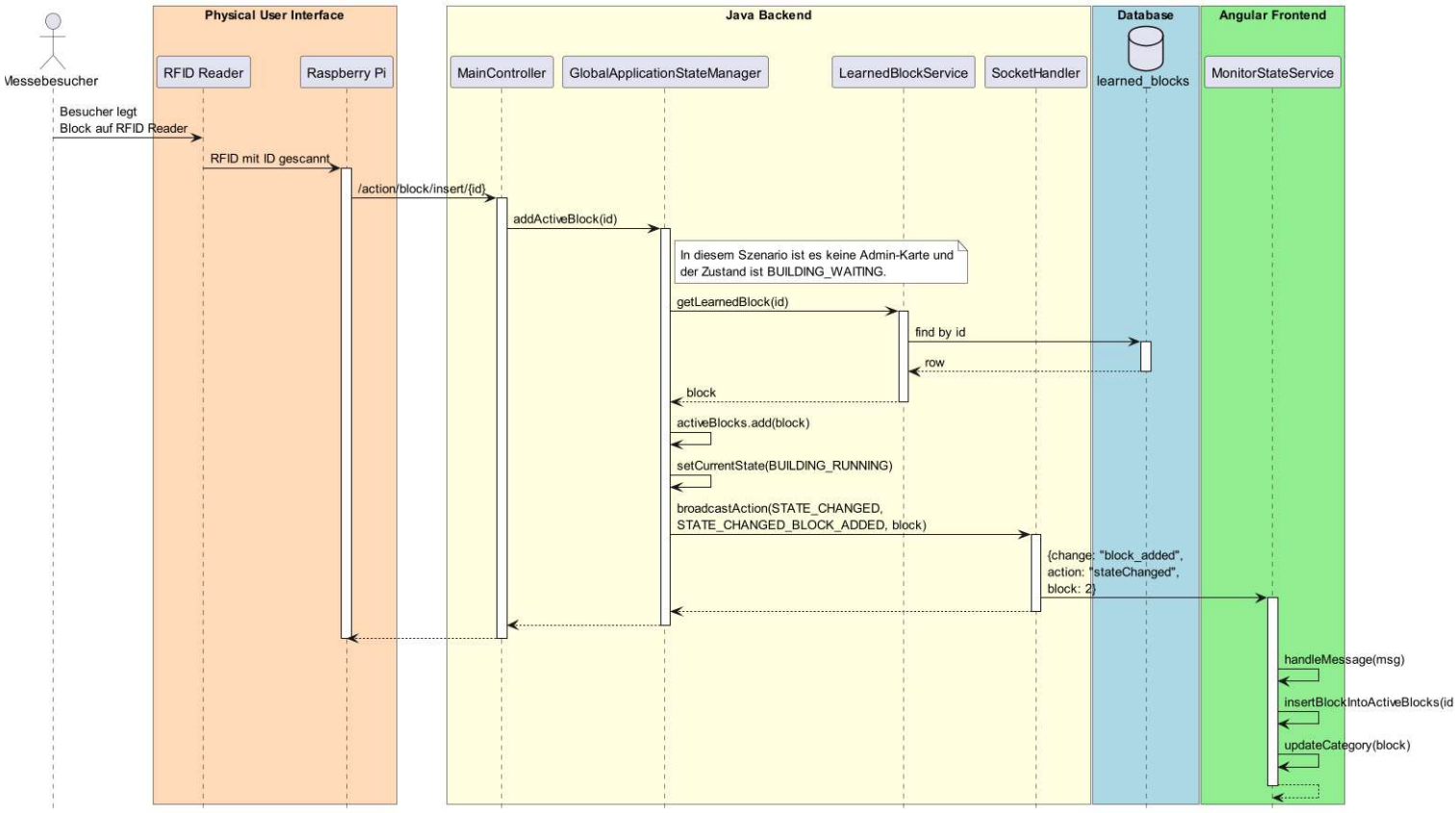


Abb. 7: Laufzeitsicht: Besucher legt Block auf Scanner

6.2. Besucher drückt Deployment Button

In diesem Ablaufdiagramm wird der Prozess dargestellt, wenn ein Besucher den Deploy-Button drückt und zuvor bereits den Deployment-Schlüssel umgelegt hat. In diesem Szenario wird davon ausgegangen, dass eine valide Konfiguration an Disketten eingelegt ist, die so “deployed” werden kann. Eine Fehlkonfiguration wird hier nicht betrachtet.

Wenn der Raspberry Pi erkennt, dass der Button gedrückt wurde, sendet er diese Information über einen HTTPS-Request an das Java Backend. Das Backend prüft den aktuellen Zustand der Anwendung und leitet je nach Situation die passenden Schritte ein. Im Folgenden alle Situationen aufgeschlüsselt, in dem betrachteten Szenario handelt es sich um Fall 3.

1. Wenn weniger als drei Disketten aktiv sind oder es sich um eine ungültige Konfiguration an Disketten handelt, bleibt das System im aktuellen Zustand, da noch keine Applikation bereitgestellt werden kann.
2. Sobald drei valide Disketten aktiv sind, aber der Schlüssel noch nicht gedreht wurde, wird der Haupt-Benutzeroberfläche mitgeteilt, dass eine entsprechende Meldung dem Besucher angezeigt werden soll.
3. Drei Disketten sind eingelegt und es handelt sich um eine valide Konfiguration.

Nun kann im Backend eine neue App-Konfiguration erstellt werden. Dies wird durch eine Insert-Anfrage an die PostgreSQL realisiert. Diese Information wird über einen

Websocket-Broadcast an das Haupt-UI weitergegeben, das nun eine Launch-Animation abspielt und einen QR-Code aus der übergebenen Applikations-Konfiguration erzeugt und anzeigt.

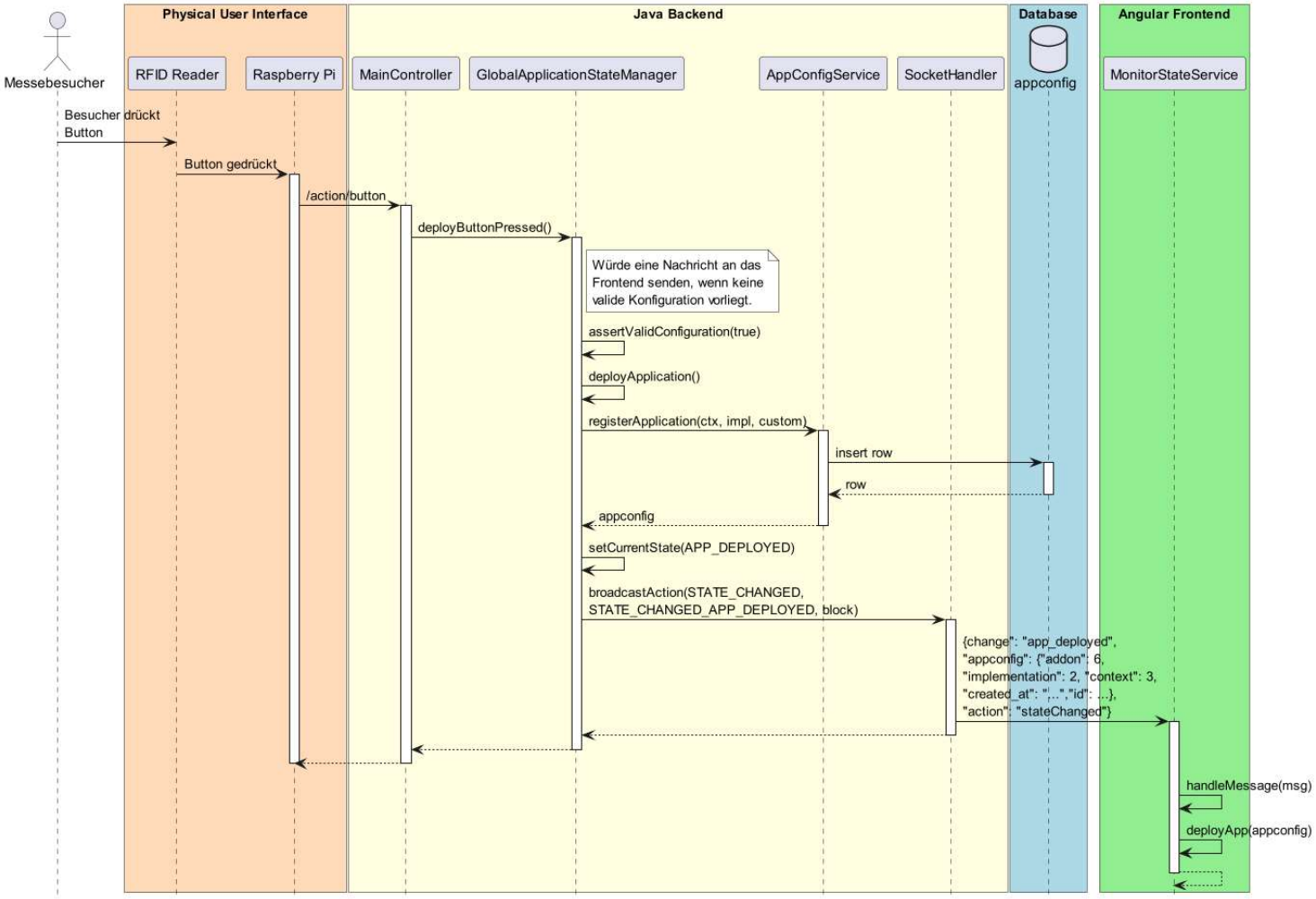


Abb. 8: Laufzeitsicht: Besucher drückt Deployment Button

6.3. Besucher ruft eine App-Konfiguration auf

In diesem Szenario scannt der Besucher den QR-Code vom Messe-Monitor, der aus einem App - Deployment resultiert. Die App Konfiguration hat dabei kein Security Addon.

Durch den Scan öffnet sich der Browser auf dem Smartphone des Besuchers und versucht, auf die entsprechende URL zu navigieren, welche der QR-Code repräsentiert. Durch den Aufruf der Route lädt der Browser die Angular Anwendung. Da die URL durch einen Route Guard, den AppConfigGuard, geschützt wird, wird dieser aufgerufen, bevor ein Öffnen der Route möglich ist. Im AppConfigGuard wird anhand der AppConfigId aus der URL die dazu passende App-Konfiguration aus dem Backend erfragt. Diese wird dann mit der Konfiguration abgeglichen, welche durch die URL widergespiegelt wird (siehe Route Zusammensetzung Ebene 3 Angular Frontend). Sollten die Konfigurationen übereinstimmen, wird diese für weitere Zugriffe im LocalStorage des Browsers gespeichert und dem Öffnen der Route stattgegeben. Nun wird innerhalb der AppConfigComponent entschieden, welche Angular Komponenten dargestellt werden, dies geschieht in Abhängigkeit der Route Parameter.

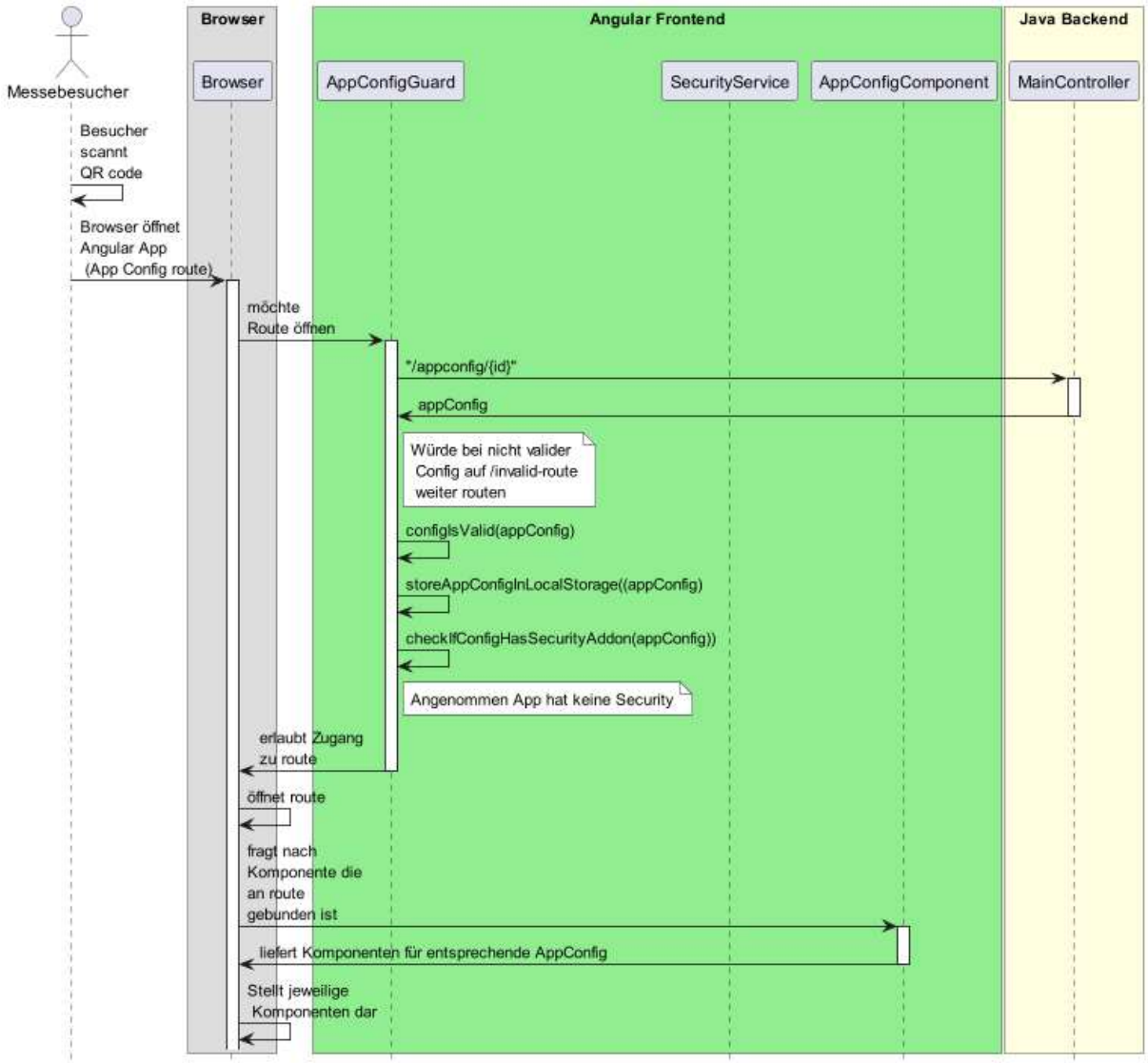


Abb. 16: Laufzeitsicht: Besucher scannt QR Code mit Smartphone - App Konfiguration ohne Security Addon

In diesem Szenario scannt der Besucher den QR-Code vom Messe-Monitor, der aus einem App - Deployment resultiert. Die App Konfiguration hat dabei das Security Addon.

Der Prozess entspricht dem aus Abbildung 16, bis zu dem Punkt, wenn die App-Konfiguration als valide bestätigt wurde. Wurde nun zusätzlich festgestellt, dass die App-Konfiguration den Security Block beinhaltet, wird geprüft, ob sich ein Benutzertoken im LocalStorage des Browsers befindet. Sollte kein Token vorhanden sein, wird der Nutzer automatisch auf die /login Route weitergeleitet, auf der er sich dann registrieren oder einloggen kann. Konnte ein Token erkannt werden, wird innerhalb des SecurityService mit Hilfe einer Backend-Anfrage geprüft, ob der Token valide ist. Ein invalider Token führt ebenso zum Reroute auf /login. Ein valider Token führt zur Erlaubnis durch den AppConfigGuard, die URL zu öffnen und der Rest des Prozesses gleicht wieder dem aus Abbildung 16.

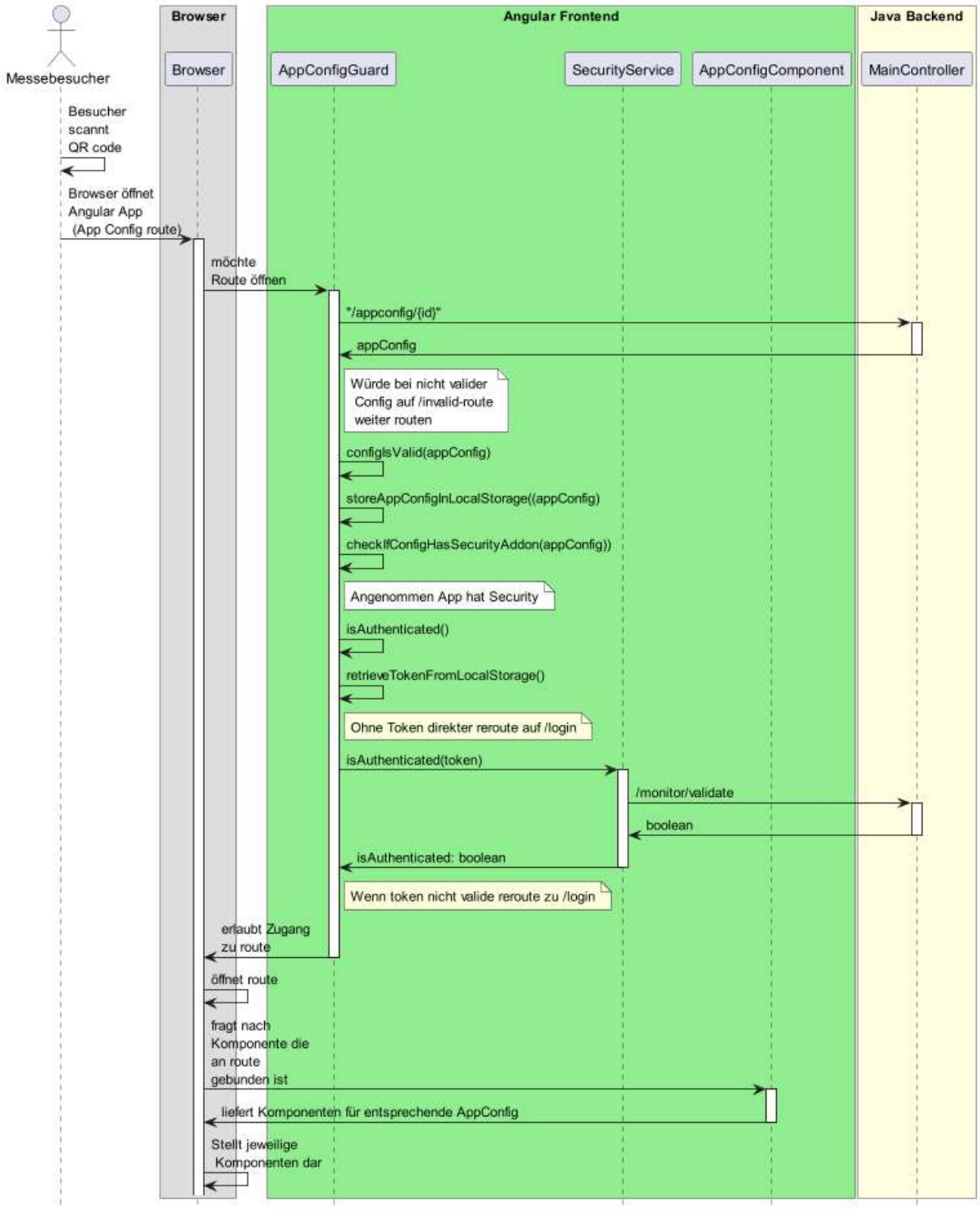


Abb. 17: Laufzeitsicht: Besucher scannt QR Code mit Smartphone - App Konfiguration mit Security Addon

6.4. Besucher bearbeitet einen Kanban-Eintrag

In diesem Szenario hat ein Messebesucher einen Kanban-Eintrag im Kanban Board, also in der Business-Planning-Security Konfiguration angelegt und gerade fertig bearbeitet. Er drückt nun auf “Speichern und Schließen”, um den Eintrag zu speichern.

Dies löst im Frontend einen Aufruf an das Backend aus, bei dem das Nutzer-Token des aktuell angemeldeten Benutzers und der aktualisierte Kanban-Eintrag mitgegeben wird. Wenn kein Security-Modul ausgewählt worden wäre, würde hier hinter den Kulissen ein Default-Token mitgegeben werden, das im Backend zu einem Default-Benutzer gemappt werden würde.

Aus dem Token wird nun die Benutzer-ID abgeleitet, indem zunächst eine Abfrage auf die token-Tabelle und danach auf die user-Tabelle gestellt wird. Diese ID wird dazu genutzt, um zu validieren, dass der Benutzer tatsächlich der Eigentümer des Eintrags ist und ihn bearbeiten darf. Dazu wird der Eintrag des Kanban-Eintrags aus der kanban-Tabelle gesucht und die Benutzer-ID verglichen.

Zuletzt wird dann noch eine Botschaft über den WebSocket Kanal an das Frontend gesendet, das eine Aktualisierung des Eintrags verursacht.

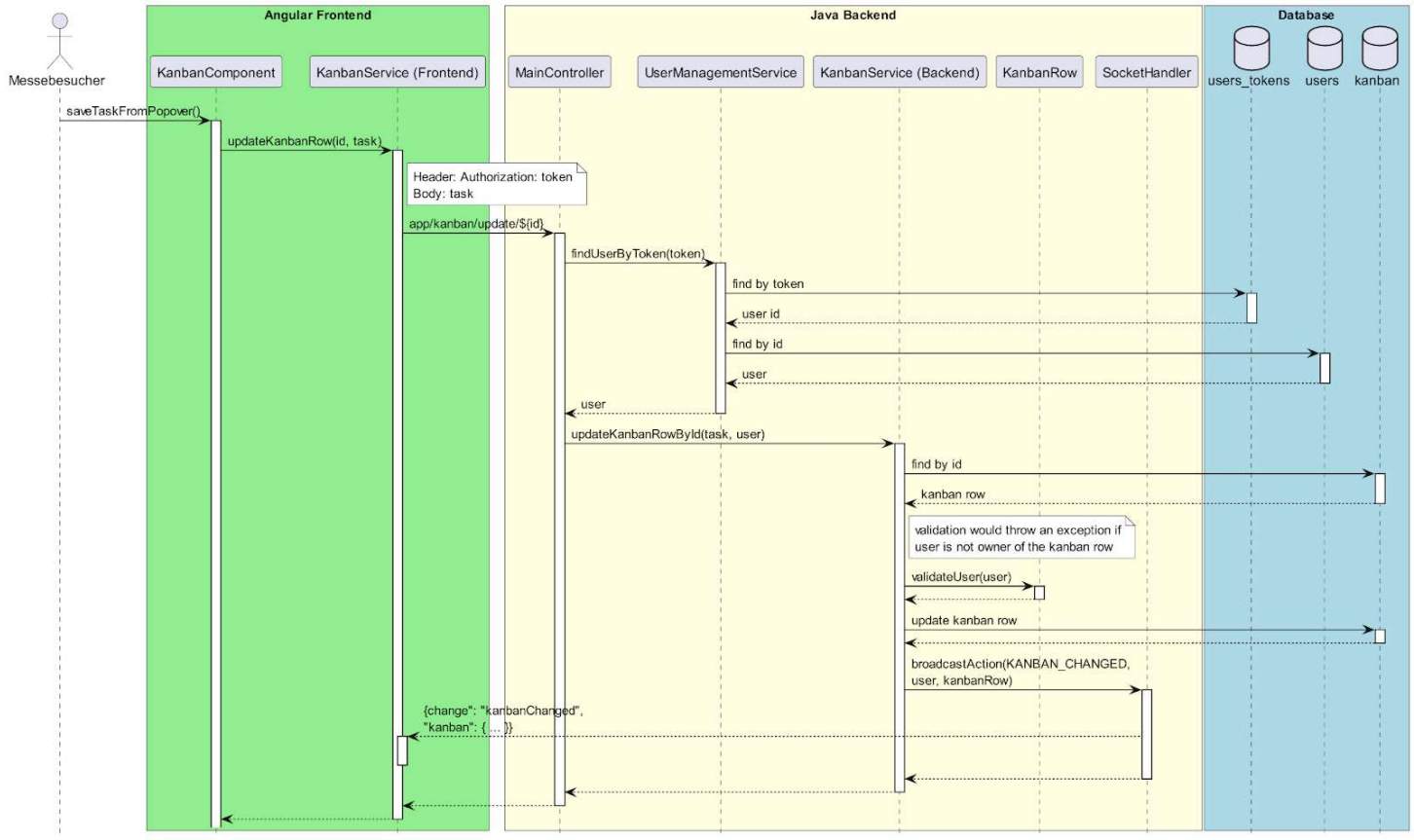


Abb. 14: Besucher bearbeitet einen Kanban-Eintrag

7. Verteilungssicht

Die Verteilungssicht (Abb. 9) 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.

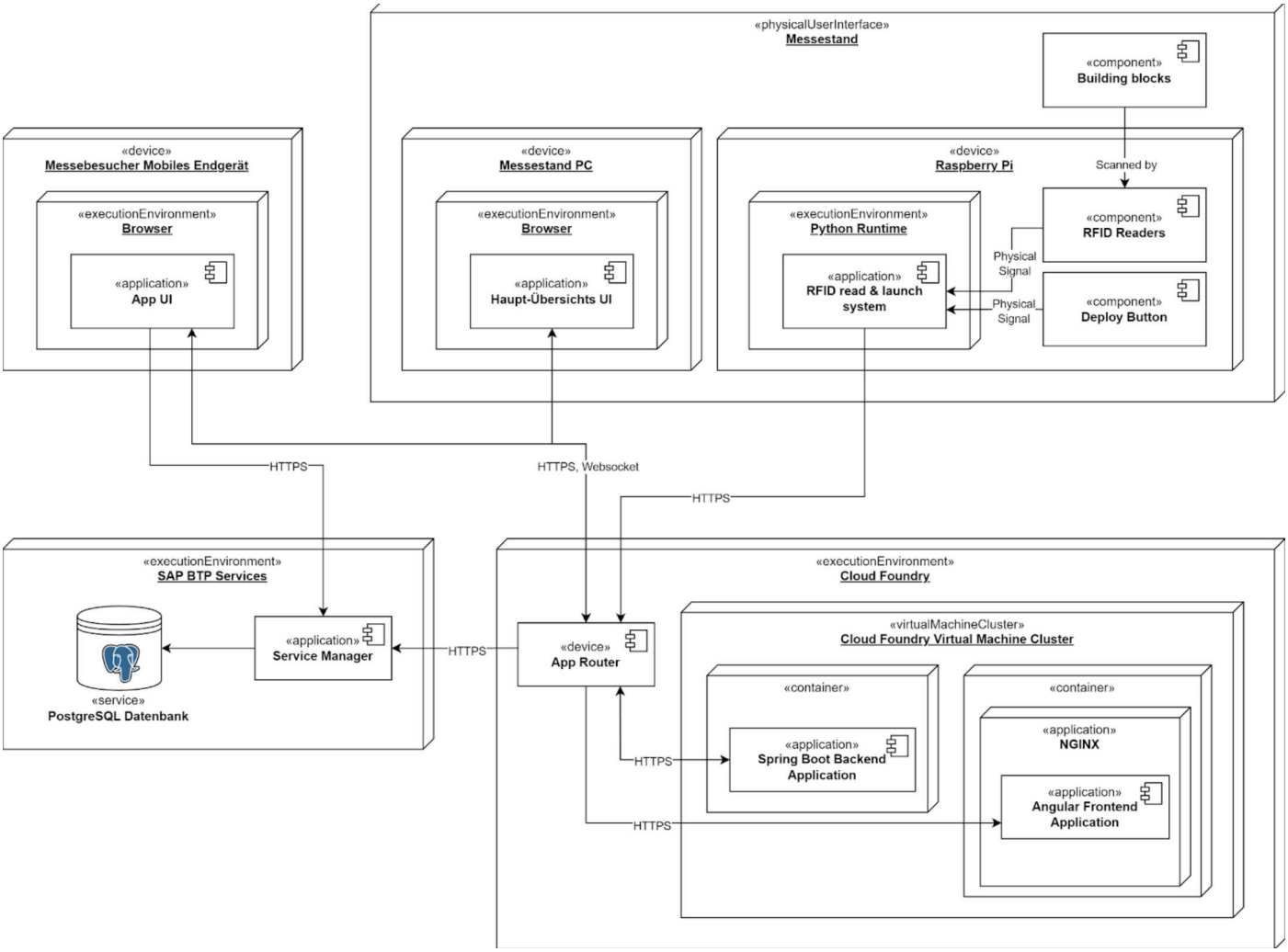


Abb. 9: Verteilungssicht

8. Architekturentscheidungen

8.1. Technologieentscheidungen

8.1.1. PostgreSQL (https://www.postgresql.org/)

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. Durch die Randbedingung RB1 wird zudem vorgegeben, dass möglichst viele Teile der Applikation auf der BTP laufen sollen.
- 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 (https://www.sap.com/products/technology-platform/hana.html)

  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 und wenige Nutzerdaten 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.

8.2. Angular (https://angular.io/)

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. Siehe RB6.
- 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** (https://react.dev/) **& VueJS** (https://vuejs.org/)
>
> ReactJS sowie Vue.js sind JavaScript-Bibliotheken zur Entwicklung von Benutzeroberflächen.
>
> - Zwar gelten ReactJS und Vue.js als leichtgewichtiger im Vergleich zu Angular, 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.

8.3. Spring Boot (https://spring.io/projects/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)
(https://www.oracle.com/java/technologies/java-ee-glance.html)

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 (https://quarkus.io/)

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.

8.4. Maven (https://maven.apache.org/)

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** (https://gradle.org/)
>
> 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.

8.5. Websockets (https://en.wikipedia.org/wiki/WebSocket)

Für unsere Anwendung nutzen wir Websockets zur Ermöglichung von bidirektionaler Kommunikation zwischen dem Server und dem Client. Bei dem Client handelt es sich 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.

8.6. Architekturmuster

8.6.1. 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.

8.6.2. 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 (https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/dao.html), 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.

Wie gesagt wird dieses Pattern bereits durch Spring umgesetzt, jedoch haben wir uns gegen die Verwendung dieser Teile des Frameworks entschieden. Wir ziehen eine eigene Implementierung dieser vor, da:

- Wir müssen die Datenbank zur Laufzeit wechseln können, was in Spring nicht ohne viel zusätzlichen Code möglich ist.
- Spring bietet viele Möglichkeiten, die wir nicht benötigen, da wir nur einzelne Abfragen nach dem Primärschlüssel benötigen, daher ist eine eigene Implementierung nicht sehr zeitintensiv.
- Bei der Recherche für diese Entscheidung wurde versucht, die Spring-Alternative in unseren Code zu Integrieren und auch nach 6 Personenstunden ist es nicht gelungen, eine Verbindung zum PostgreSQL Datenbank-Service auf der SAP BTP herzustellen, daher haben wir diesen Ansatz recht schnell aufgegeben.

8.6.3. 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.

9. 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.

9.1. Q1: Fehlertoleranz Software

Das System ist, bei nicht vorgesehener Nutzung der Anwendung, in der Lage, Fehler adäquat zu behandeln und den Nutzer auf Fehleingaben hinzuweisen. Hierbei bleibt das System zu jeder Zeit lauffähig.

Zur Erreichung dieser Anforderung wurde im Backend sowie im Frontend ein umfangreicher Error Handling Mechanismus eingebaut, welcher Fehleingaben, welche das System in seiner Lauffähigkeit beeinträchtigen könnten, vollständig vermeidet. Nicht vorgesehene Nutzung, die aufgrund unserer Hardwarearchitektur nicht vollständig vermieden werden kann, wird von unserer Anwendung verarbeitet und bietet dem Nutzer Hinweise, um erfolgreich mit unserer Anwendung zu interagieren. Hierbei wird die Lauffähigkeit nicht gefährdet.

9.2. Q2: 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.

Die Erreichbarkeit unserer Applikationen hängt vollständig von der Infrastruktur der SAP BTP ab. Hierbei verlassen wir uns auf die Zuverlässigkeit und Ausfallsicherheit der SAP Platform.

Die Neustart Dauer unserer Applikation liegt deutlich unter 5 Minuten und kann innerhalb des SAP BTP Cockpits angestoßen werden.

10. Glossar

Begriff	Erklärung	Quelle
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.	https://de.wikipedia.org/wiki/RFID
RFID-Tags	RFID-Tags sind eine Art Verfolgungssystem, das zur Identifizierung von Artikeln mithilfe einer Art smartem Barcode verwendet wird	https://ioxlab.de/de/iot-tech-blog/was-sind-rfid-tags/
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.	https://rfid-finder.com/rfid-auto-id-technologie/rfid-reader/#:~:text=Ein%20RFID%2DReader%20ist%20ein,Beh%C3%A4lter%2C%20Material%20und%20Werkzeugen%20auszulesen.
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	https://de.wikipedia.org/wiki/Programmierschnittstelle
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.	https://de.wikipedia.org/wiki/Java_Database_Connectivity
DevOps	Unter DevOps versteht man diverse Praktiken und Tools, die die Prozesse zwischen Softwareentwicklungs- und IT-Teams automatisieren und integrieren.	https://www.atlassian.com/de/devops#:~:text=Unter%20DevOps%20versteht%20man%20diverse,Kommunikation%20und%20Zusammenarbeit%20sowie%20Technologieautomatisierung.

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). Continuous Integration: Entwickler führen regelmäßig Codeänderungen zusammen, um Konflikte frühzeitig zu erkennen. Automatisierte Tests helfen bei der Fehlererkennung. Continuous Deployment: Code Änderungen werden automatisch in die Produktion übertragen, nachdem sie erfolgreich getestet wurden.	https://www.redhat.com/de/topics/devops/what-is-ci-cd
CI/CD Pipeline	Eine CI/CD-Pipeline automatisiert mehrere Schritte, die zur Bereitstellung einer neuen Softwareversion ausgeführt werden müssen. CI/CD-Pipelines verkörpern eine Praxis zur Verbesserung der Softwarebereitstellung.	https://www.redhat.com/de/topics/devops/what-cicd-pipeline
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.	https://www.sap.com/germany/products/technology-platform/what-is-sap-business-technology-platform.html
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.	https://www.cloudfoundry.org/
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.	https://de.wikipedia.org/wiki/Virtuelle_Maschine
Container	Container sind Softwarepakete, die alle Elemente enthalten, die zur Ausführung in beliebigen Umgebungen erforderlich sind. Container virtualisieren dadurch das Betriebssystem.	https://cloud.google.com/learn/what-are-containers?hl=de#:~:text=Container%20sind%20Softwarepakete%2C%20die%20alle,dem%20pr

		ivaten%20Laptop%20eines%20Entwicklers.
REST	Eine REST-API (Representational State Transfer Application Programming Interface) ermöglicht den Austausch von Daten zwischen Anwendungen über das HTTP-Protokoll. REST basiert auf einem Architekturstil, der bestimmte Prinzipien betont. Dazu gehören die Verwendung der standardmäßigen HTTP-Methoden wie GET, POST, PUT und DELETE für den Zugriff auf Ressourcen sowie einheitliche und eindeutige Bezeichnungen (URIs) für diese Ressourcen. Durch die Nutzung von Repräsentationen können Daten in verschiedenen Formaten wie JSON oder XML übertragen werden.	https://www.redhat.com/en/topics/api/what-is-a-rest-api