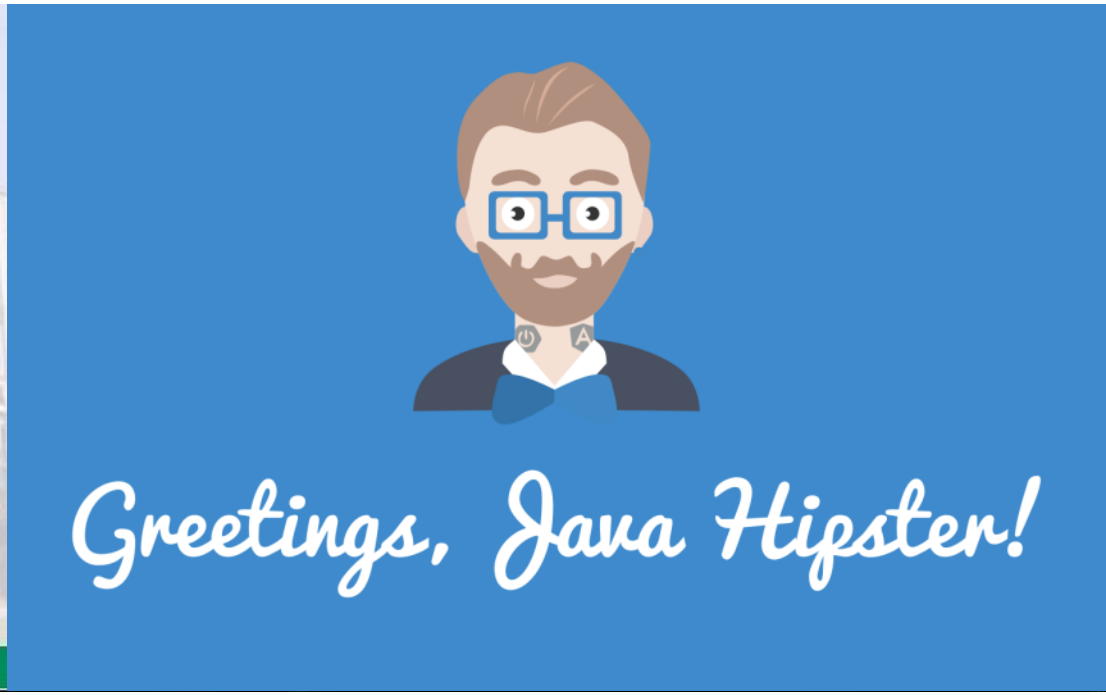
anderScore
trust in competence

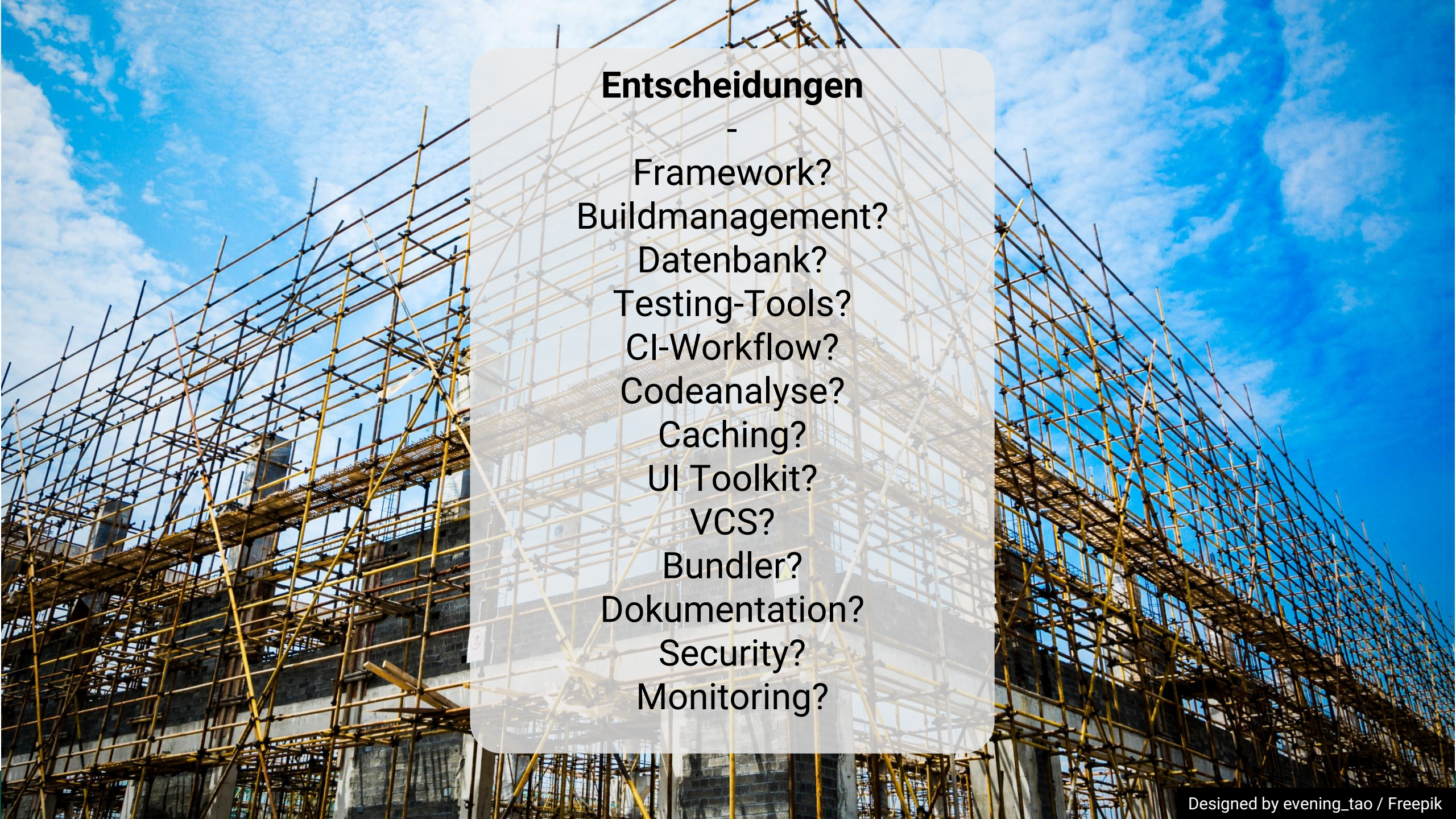
JHipster

Spring mit modernen Web Frontends



09.10.18, Max Johenneken


Neue Web Anwendung mit
Java soll her
-
Womit fangen wir an?
Designed by awesomecontent / Freepik


Entscheidungen
-
Framework?
Buildmanagement?
Datenbank?
Testing-Tools?
CI-Workflow?
Codeanalyse?
Caching?
UI Toolkit?
VCS?
Bundler?
Dokumentation?
Security?
Monitoring?
Designed by evening_tao / Freepik

Unsere Agenda

- Dauer: ca. 2 Stunden bis 19:30Uhr
- JHipster verstehen und anwenden
- Livedemo monolithische Anwendung generieren
- Die JHipster „Micro“service Architektur
- Showcase: Architektur Projekt AutoLiefer mit JHipster

Code und Slides auf Github:

https://github.com/goldschmiede/2018-10-12-Jhipster

https://jhipster.tech

max.johenneken@anderscore.com

Moderne Web Apps

Anspruch der Benutzer ist gestiegen:

- Schönes Design
- Kurze Ladezeiten
- Dynamische Seitenaktualisierung
- ➔ Nutzen der neusten HTML/CSS/JS Technologien ist Voraussetzung für gute UX

Auch der Entwickler ist Nutzer

- Wir wollen die Software schnell bauen können
- Schnelles Feedback beim Entwickeln
- ➔ Designänderungen - Live Reload
- ➔ Feedback vom Kunden – Kurze Zyklen Continuous Deployment
- ➔ Backend ändern ohne Neustart – Hot Reloading
- ➔ Tests ausführen

- Wir brauchen dafür die richtigen Tools!

Systeme bereit für den Ernstfall

- Web Anwendungen sollen auch unter große Mengen paralleler Zugriffe alle Anfragen bearbeiten.

- ➔ Wir brauchen robuste, skalierbare und performante Software

Ziele von JHipster

anderScore trust in competence

- Bereitstellung eines hochperformanten Java Stacks mit Spring Boot auf der Serverseite und einer Auswahl aus modernen DB-Technologien.
- Einer schicken, modernen, responsiven UI mit Angular oder React und Bootstrap.
- Einer robusten Microservice Architektur mit der JHipster Registry, dem Netflix OSS Stack, dem ELK Stack und Docker.
- Einem mächtigen Buildmangement Workflow mit Yeoman, Webpack und Maven oder Gradle.

Konzepte von JHipster

- Scaffholding(engl. Gerüst) von Anwendungen
  - Die Anwendung wird Initial aufgesetzt
  - Wichtige Designentscheidungen sind wählbar
  - Upgrading ist bedingt möglich
- Der erzeugte Code ist qualitativ hochwertig
  - Enthält keine bekannten Sicherheitslücken
  - Ist mit ausreichend Abdeckung getestet
  - Wartbarkeit, Anpassbarkeit sind zentrale Qualitätsmerkmale
  - Sonar JHipster Sample App
- Yeoman Generator
  - Werkzeug zum Erstellen von Generatoren
  - Texttemplates (Bsp. Nächste Folie)

Beispiel Template Service Klasse

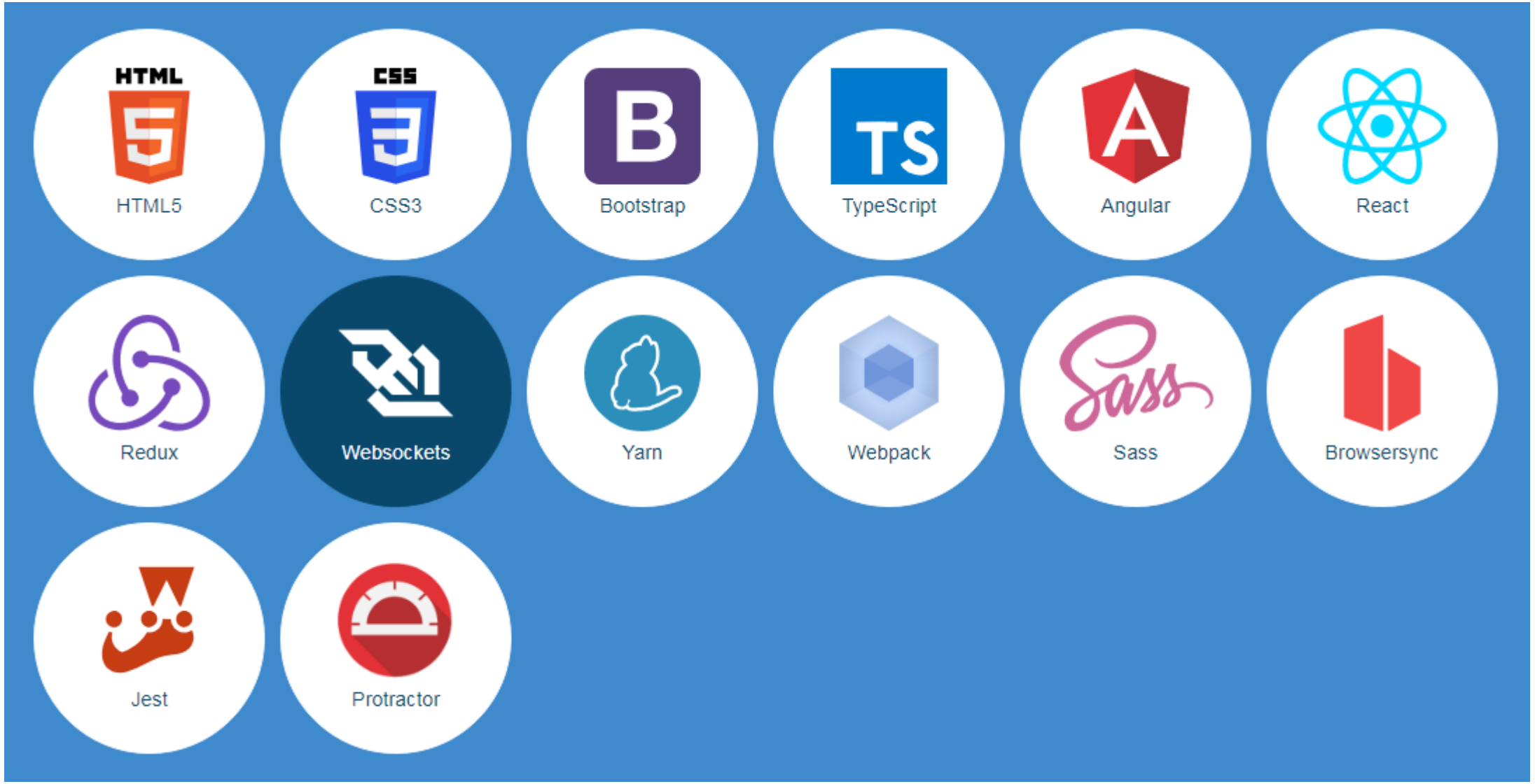
anderScore trust in competence

```
<% if (useInterface === false) { %>import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;<% if (databaseType === 'sql') { %>
import org.springframework.transaction.annotation.Transactional;<% } %>

@Service<% if (databaseType === 'sql') { %>
@Transactional<% } %>
public class <%= serviceClass %> {

    private final Logger log = LoggerFactory.getLogger(<%= serviceClass %>.class);

}<% } else  { %>public interface <%= serviceClass %> {

}<% } %>
```

Zusätzliche Features

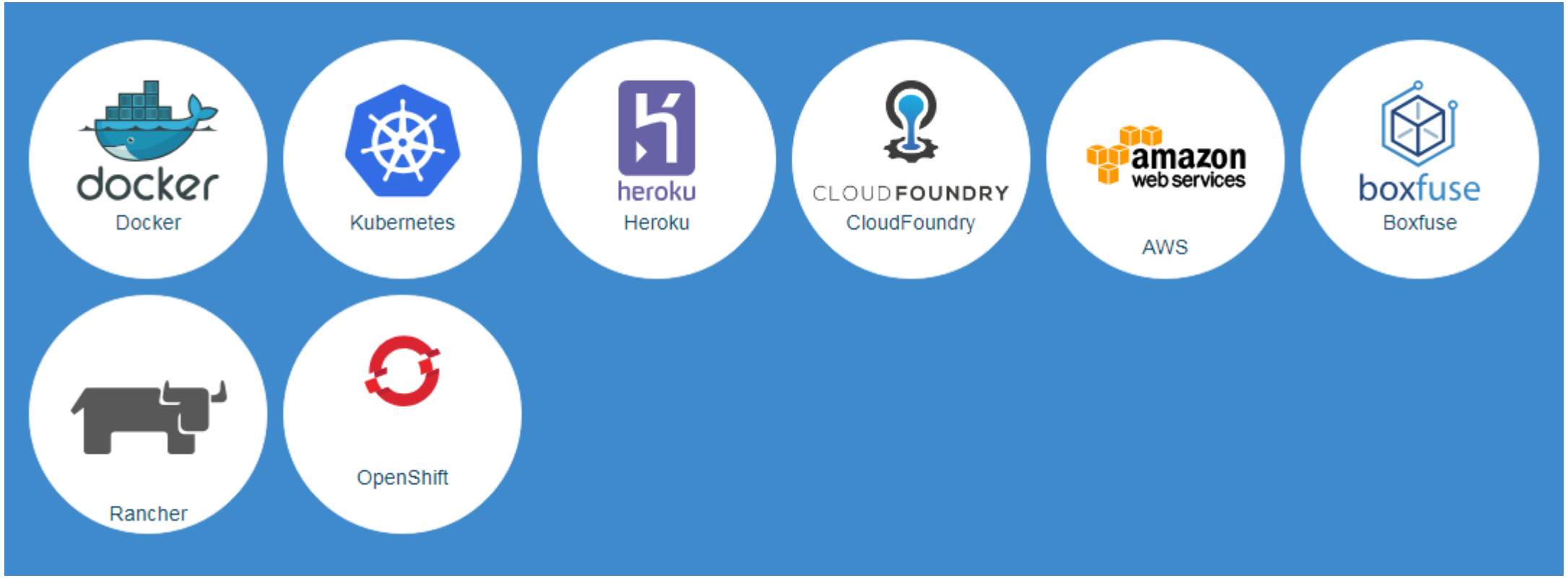
- Entities generieren mit JDL (JHipster Domain Language) oder XMI
  - **C**reate, **R**ead, **E**dit, **D**elete, **O**verview (CREDO) Operationen
  - DB Tabellen, Entity, ggf. DTO
  - Service, Repository, RESTController
  - Angular Service
  - Angular Komponenten
  - Unit- und Integration-Tests für JavaScript und Java
  - Konfiguration von Security(Endpoints), Caching, …
- Spring Services generieren
- Continuous Integration/Deployment Pipeline generieren
- Deployment Konfiguration generieren

Frontend Technologien

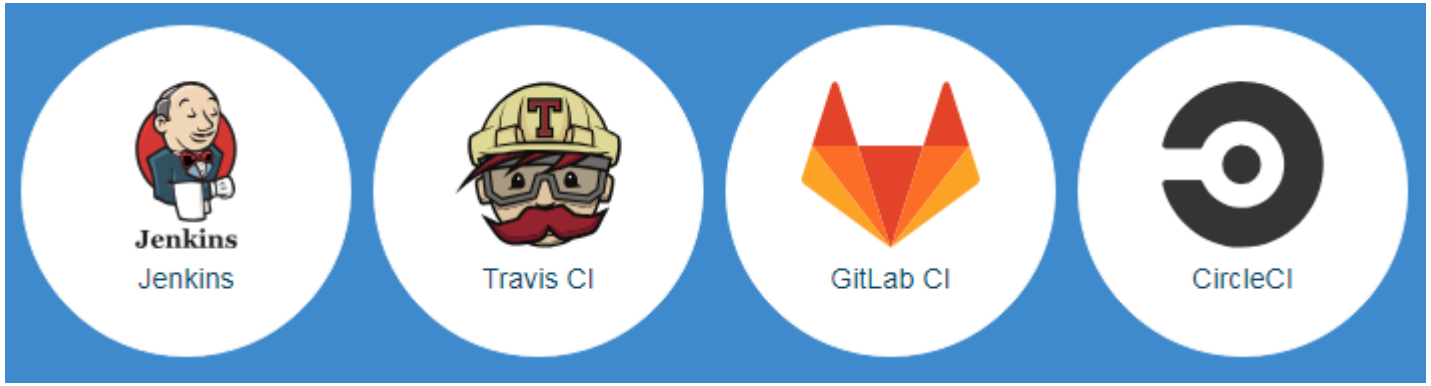


Backend Technologien

Deployment Optionen

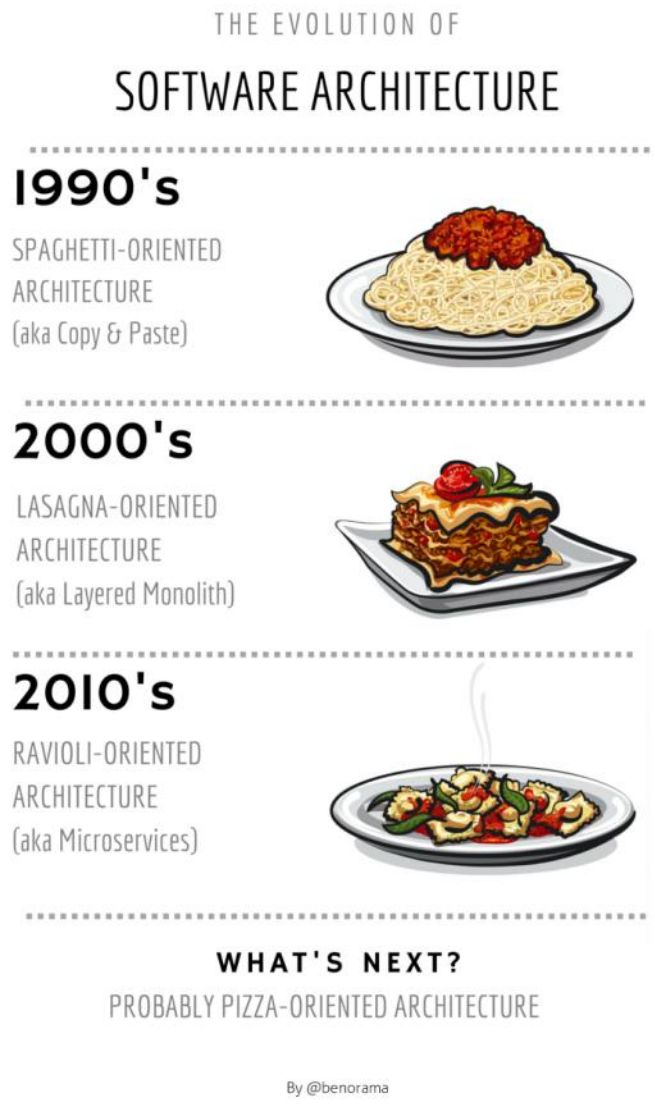


CI/CD Optionen



Livedemo:
Artikelverwaltung mit JHipster entwickeln

Mircoservices mit JHipster



Microservice - Definitionen

- „Microservices are small, autonomous services that work together“ (Newmann, 2015)

- „Microservices sind unabhängig deploybare Module“ (Wolff, 2018)

- „Ein Microservice ist ein isolierter Dienst mit eigener Laufzeitumgebung und NonShared Storage State. Er hat nur eine einzige Geschäftsaufgabe, aber erledigt diese
gut.“ (Takai, 2017)
  - Erfüllung des Single-Responsibility-Prinzip (SOLID)

- Microservices sind primär ein organisatorisches Muster und nur zum (kleinen) Teil ein Architekturmuster” (Starke, 2017)

Microservice - Eigenschaften und Aufbau

anderScore trust in competence

- Jeder Microservice hat eine eigene Laufzeitumgebung
- Jeder Microservice hat sein eigenes Domänenmodell(Bounded Context) (Evans, 2004; Wolff, 2016)
- Kommunikation über das Netzwerk z.B. nach REST mit HTTP
- Flexibles Einsetzen in schlanke Virtuelle Maschinen oder Container
- Bereitstellung in Public/Private-Cloud Umgebungen mit Techniken zur Skalierung und Robustheit pro Service
- Jeder Microservice hat eigene Datenhaltung d.h. keine Integration über DB (Polyglot Persistence)
- Größe eines Microservice?

In Anlehnung an Alda, 2018

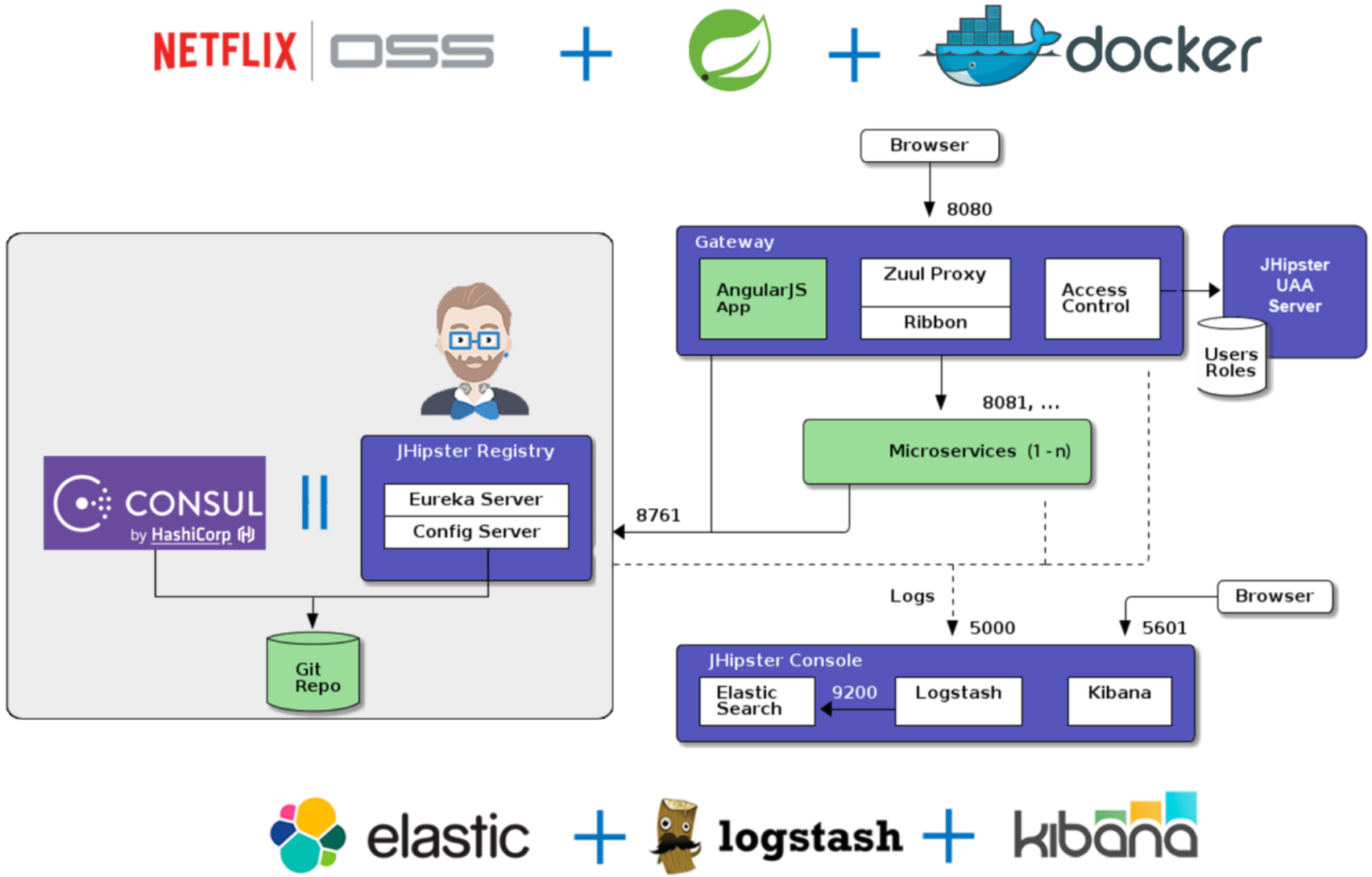
Microservice – Größe eine Microservice

„Bei genauerer Betrachtung ist die Größe aber **gar nicht so wichtig**.

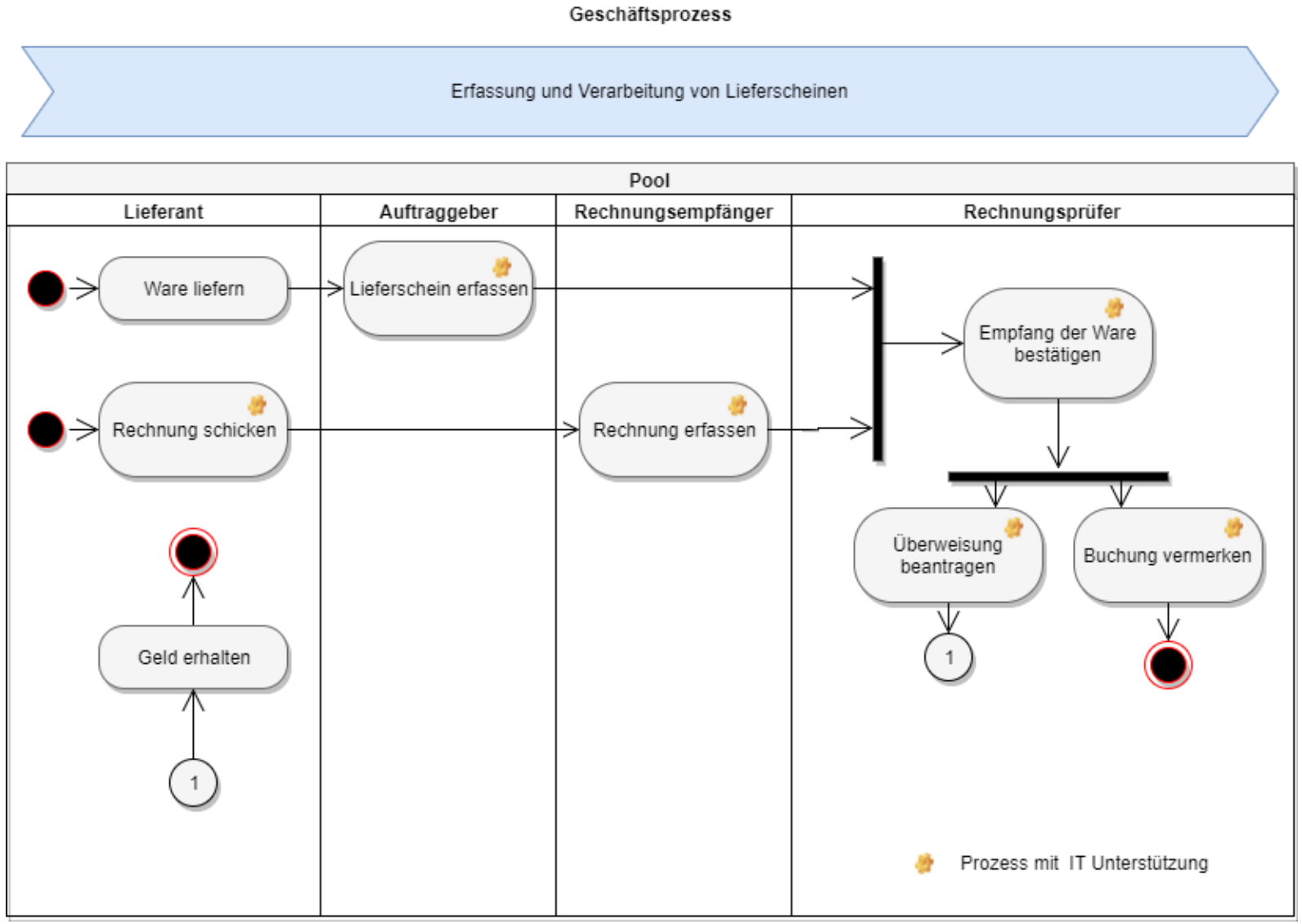
Die Teamgröße, die Modularisierung und die Ersetzbarkeit der Microservices legen jeweils eine **obere Grenze** fest.

Die **untere Grenze** kommt von den Transaktionen, der Konsistenz und der verteilten Kommunikation.“(Wolff, 2018)

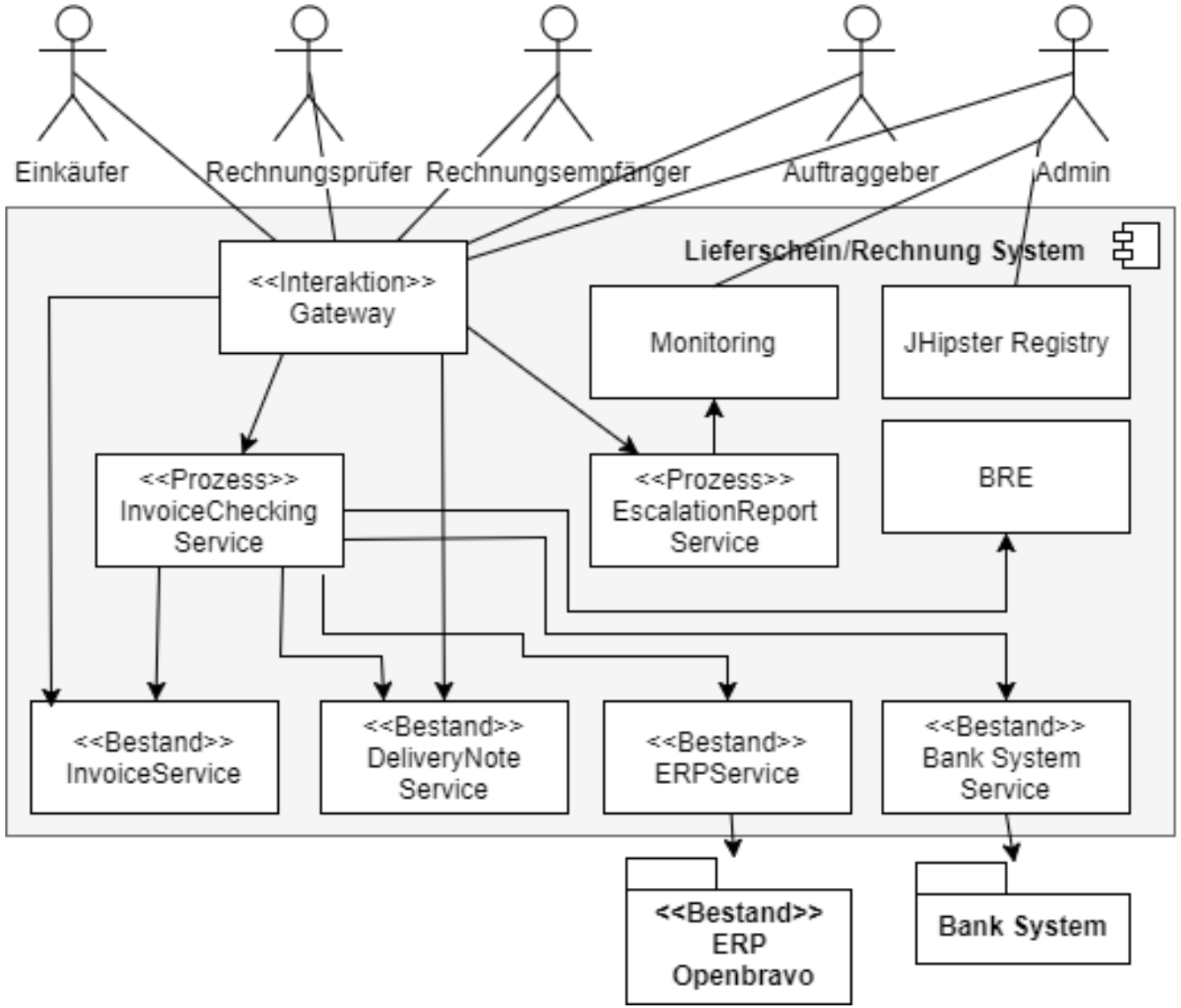
JHipster Microservices Architektur



Showcase Projekt AutoLiefer



Showcase Projekt AutoLiefer



Showcase:

Projekt AutoLiefer

Erfassung und Verarbeitung von Lieferscheinen/Rechnungen

Ausblick – Weitere Funktionen

- Blueprints & Module
  - Erweiterung des generierten Codes durch Module
  - Ändern bestehender Dateien über Blueprints
  - JHipster Marketplace
  - z.B. Kotlin statt Java, Ionic im Frontend

- Upgrading von Anwendungen
  - Automatische Prüfung auf Updates
  - Generator nutzt einen Git Branch

- Jhipster Online
  - Jhipster Generator als Web App
  - Kann Projekte auf Github/Gitlab bzw. als *.zip generieren

Fazit

- Wann lohnt sich der Einsatz von JHipster?
  - CRUD zentrierte Anwendungen
  - Prototyping
  - Evaluation/Vergleich verschiedener Technolgie-Stacks
- Für was kann ich JHipster besonders gut einsetzen?
  - Neue Technologien in Anwendung kennen lernen
  - Aufwand für Scaffholding von Projekten verringern
  - Als „Nachschlagewerk“ für die Kombination verschiedener Technologien
- Wann sollte man JHipster nicht einsetzen?
  - Weiterentwicklung bestehender Software
  - Mehr Aufwand den Code zu anzupassen als in selbst zu schreiben
  - Entwicklungsteam ist nicht ausreichend vertraut mit den eingesetzten Technologien

Diskussion

- Eure Fragen und Anmerkungen?

Hier ein paar Denkanstöße:

- Was müsste man technisch oder organisatorisch in deinem Projekt ändern, um JHipster einzusetzen?
- Sind Generatoren die „Zukunft“?
- Klassen generieren vs. generische Klassen implementieren?
  - Änderbarkeit?
  - Wartbarkeit?
- Was könnte man noch sinnvolles in einen Generator auslagern?
  - Aggregationen? z.B. Summe aller Rechnungsbeiträge
  - Zustandsübergänge? z.B. Eskalation eines Vorgangs, Abschließen
  - …weiteres?

Literatur

- JHipster Dokumentation – https://jhipster.tech
- Eberhard Wolff, 2018, Microservices – Grundlagen flexibler Software Architekturen
- Gernot Starke, 2018, Effektive Softwarearchitekturen – Ein praktischer Leitfaden
- Daniel Takai, 2017, Architektur für Websysteme