Wann sich der Applikations-Generator JHipster lohnt - und wann nicht

BETTER PROJECTS
FASTER

Karsten Silz

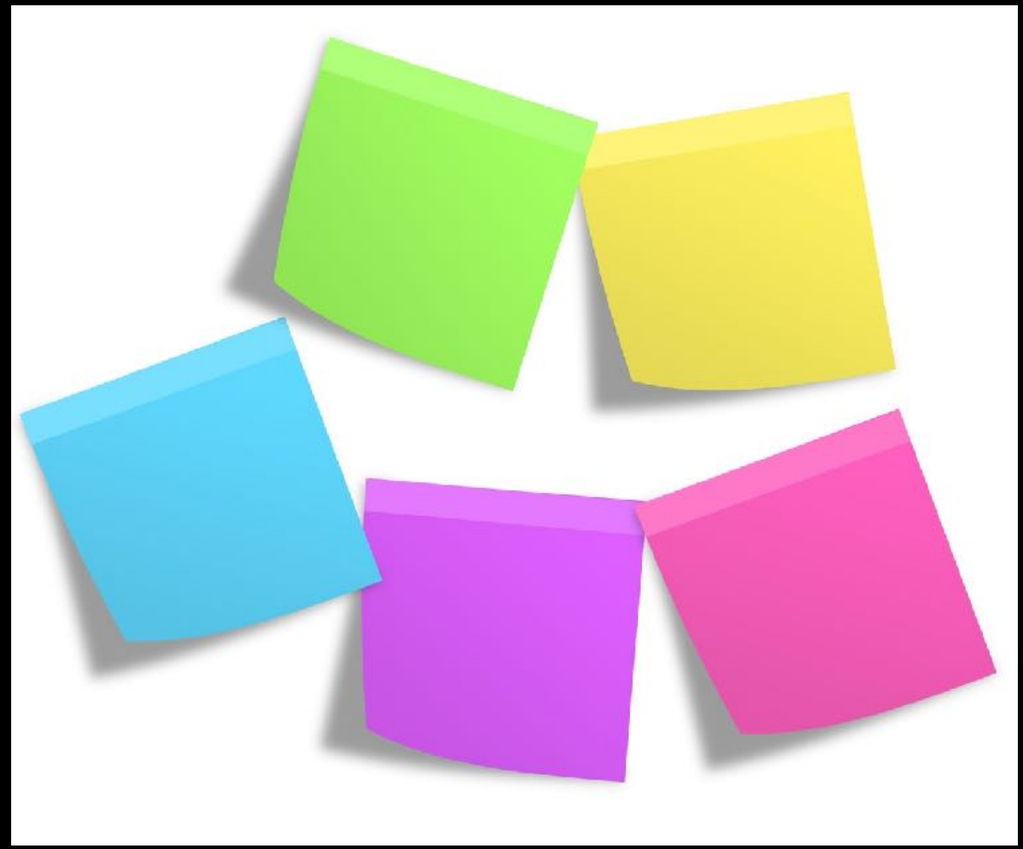
Java Forum Stuttgart
4. Juli 2019

Warum JHipster?

JHipster im Detail

Demo: "Online-Shop"

- Komplett mit JHipster generiert
- Continuous Integration, Tests, Docker & Deployment in die Cloud



JHipster liefert schneller bessere Java-Projekte, weil es beim Programmieren, im Projekt und beim Lernen Zeit spart.

Warum sollten Sie mir glauben?
Will ich was verkaufen?

Java-Entwickler seit 20 Jahren

13 Jahre Entwicklungsleiter für Software-Produkt in US-Startup

Als Freiberufler viele Java-Projekte, Prozesse & Tools aufgesetzt & bewertet

1 JHipster-Projekt mit Angular & Docker in Produktion geführt, 1 weiteres in der Entwicklung

Ich würde mich freuen, wenn Sie meiner Mailing List zu JHipster und Docker beitreten würden


JHipster-Projekt
generieren

Was haben Sie gerade gesehen?

JHipster generiert Angular & Spring Boot Applikation mit unserem Datenmodell

Git-Projekt wird auf Gitlab hochgeladen

Continuous Integration für Gitlab mit JHipster generiert & gestartet

Applikation lokal gestartet

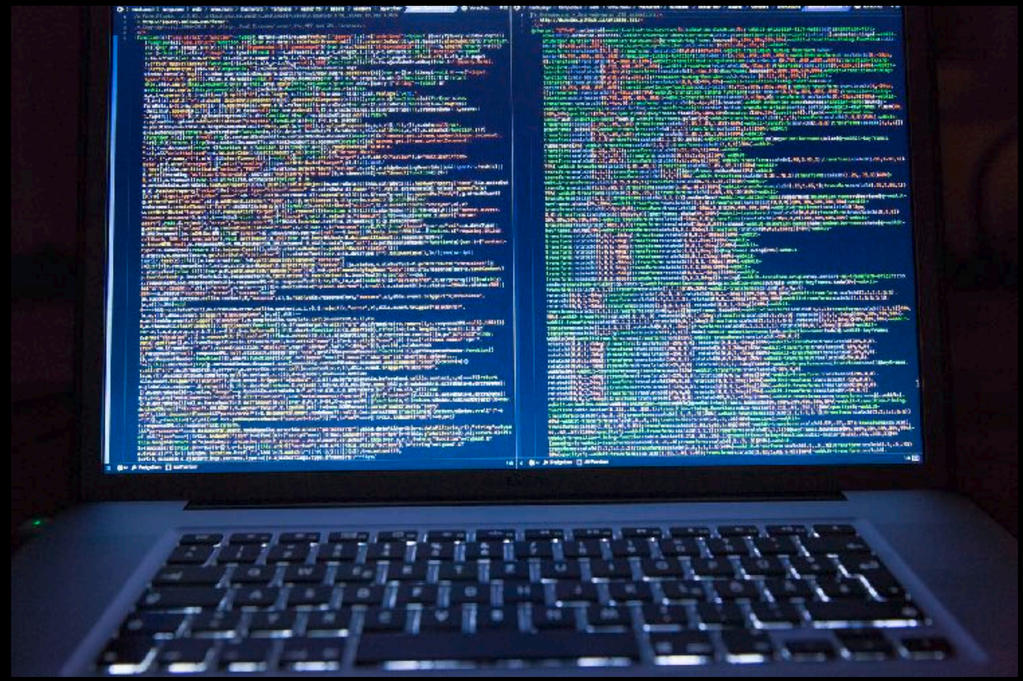
Stellen Sie sich vor:
Ihr Chef hat eine Aufgabe für Sie!

Sie

Java-Profi

Seit 5 Jahren in JEE-Projekt
(JSF, WebSphere & Oracle)

Alle vier Monate ein Major
Release im Projekt



Ihr Chef will

Single Page Application (SPA)
mit JavaScript

Open Source

Container & Cloud

Agilität & DevOps

Major Release alle zwei Wochen

Container

Server wie eine JAR-Datei programmieren, verteilen und aktualisieren

Gleiche "Server" in Dev, Test & Produktion

DevOps

Wir Entwickler sind mehr für Test & Betrieb zuständig

Mehr Automatisierung bei Build, Test & Deployment

Bessere Diagnostik zur Laufzeit

Mehr & bessere Tests

Wie SPA liefern?

Unit Tests?

Continuous Integration?

User Roles in JavaScript?

Responsive Layout?

Build Tool?

SPA Framework?

Browser Tests?

Login mit JavaScript?

JavaScript im Java-Projekt?

Sie brauchen geschätzt **4 Wochen** für den Prototyp

Ständige Bewegung im Projekt - gerade im JavaScript-Bereich: Security Patches, Updates, neue Frameworks & Libraries

Deswegen für Projekt-Updates: **1 Woche** pro Quartal

Erstes Jahr: 4 + 3x1 = **7 Wochen** für Erstellung & Maintenance des JavaScript-Projekts

Mit JHipster (geschätzt):
Nur 1 Woche statt 7 Wochen!

Wie macht JHipster das?!

Open Source auf Github: https://www.jhipster.tech

Erzeugt **produktionsreifen Code** nach **Best Practices**

Generiert Applikationen & Projekte: einheitliches Build-System, Unterstützung für Continuous Integration & Cloud, Tests, Nutzer-Verwaltung und Administration

Liefert regelmäßige Updates für bestehende Applikationen: Sicherheits-Updates, neue Frameworks- & Bibliotheks-Versionen, Austausch von Frameworks & Bibliotheken

Erzeugt CRUD Screens: JHipster generiert Back End und Front End für projekt-spezifisches Datenmodell in JHipster Domain Language (JDL)

JHipster Domain Language (JDL) im Detail

```
application {
  config {
    applicationType monolith
    authenticationType jwt
    buildTool gradle
    prodDatabaseType postgresql
    clientFramework angularX
    nativeLanguage en
    languages [en, de]
  }
}
deployment {
  deploymentType docker-compose
}
```

```
entity Product {
  name String required unique minlength(2) maxlength(90)
  price Float required min(0)
  description TextBlob required
  picture ImageBlob required
  specification Blob
  category ProductCategory
}

enum ProductCategory {
  Laptop, Desktop, Phone, Tablet, Accessory
}
```

```
relationship OneToOne {
  Shipment{order(name) required} to
      ShoppingOrder{shipment(shippedAt)}
}

relationship OneToMany {
  Product to ProductOrder{product(name) required}
}

service * with serviceImpl
dto * with mapstruct
```

JHipster-Projekt Demo

Was haben Sie gerade gesehen?

Applikation mit Nutzer-Verwaltung, Administration und CRUD Screens für unser Datenmodell

Continuous Integration at GitLab

Responsive Layout für mobile Geräte

Docker Image generiert

Deployment auf Docker Hub gestartet

JHipster im Detail

Architektur: Monolith oder Microservices (Eureka Server, Consul)

Persistenz: SQL (JPA + Hibernate + Liquibase: H2, MySQL, MariaDB, PostgreSQL, Oracle, SQL Server) oder NoSQL (MongoDB, Cassandra, Couchbase)

Back End: Spring Boot mit Profilen `dev` & `prod` und Java 8 oder Java 11 oder Kotlin (Beta)

Front End: Angular oder React oder Vue.js (Beta)

Build-System: 1 Git-Projekt mit Maven oder Gradle

Deployment: "Executable JAR-Datei" mit SPA & Servlet Container

Unit Tests: Generiert mit jUnit (Java) und Jest + Jasmine (Javascript)

Integration Tests: Generiert mit jUnit (Java)

Acceptance Tests: Browser-Tests generiert mit Protractor, Cucumber für BDD-Tests nur konfiguriert

Load Tests: Konfiguriert für Gatling

Continuous Integration: Konfiguriert Jenkins, Travis, GitLab & Azure Pipelines

Cloud: Konfiguriert Kubernetes, AWS, Google App Engine, Cloud Foundry, Heroku, OpenShift (Beta)

I18N: Unterstützt 40+ Sprachen

Layout: Bootstrap & Font Awesome

Application Messaging: Spring Websocket oder Apache Kafka

Full Text Search: Elasticsearch

Authentication & Authorization: Spring Security mit HTTP Session oder JWT oder OAuth 2.0/ OIDC, User Roles `ADMIN` & `USER`, Nutzer-Verwaltung, Login-Audit, Selbst-Registrierung, Passwort-Reset

Administration: Application Metrics, Konfigurations-Details, REST-API-Details

CRUD für Ihr Datenmodell: Übersichts-Tabelle mit Paging, Detail-Ansicht, kombiniertes Formular für "Create & Update", Form Validation, Uploads & Downloads für Bilder und Dateien

Welcher Code wird für CRUD generiert?

Java

- Liquibase-Definitionen
- JPA Entity (mit Cache)
- Spring JPA Repository
- DTO & Mapper
- Service-Klasse
- REST-API

Angular

- Typescript DTO
- 1 Modul pro Entity
- Client-Service für REST-API
- 4 Komponenten für CRUD
- Lokalisations-Dateien


Docker &
Browser Tests

Was haben Sie gerade gesehen?

Browser-Tests mit Protractor

Docker Image zu Docker Hub gepusht

Deployment auf Azure gestartet

Sie müssen nicht alle Features von JHipster nutzen!

Eigenes Datenbank-Schema: Generierte Liquibase-Dateien überschreiben oder ersetzen (z.B. durch SQL-Script), JPA-Mapping ändern

Eigene Nutzer-Verwaltung: Generierte Nutzer-Verwaltung abschalten, eigenes Nutzer-Modell mit Spring Security einsetzen

Kein Front End für Ihr Datenmodell: Wird Pro Entity in JDL konfiguriert

Gar kein CRUD für Ihr Datenmodell: Back End und Front End ohne JDL komplett "per Hand" erzeugen

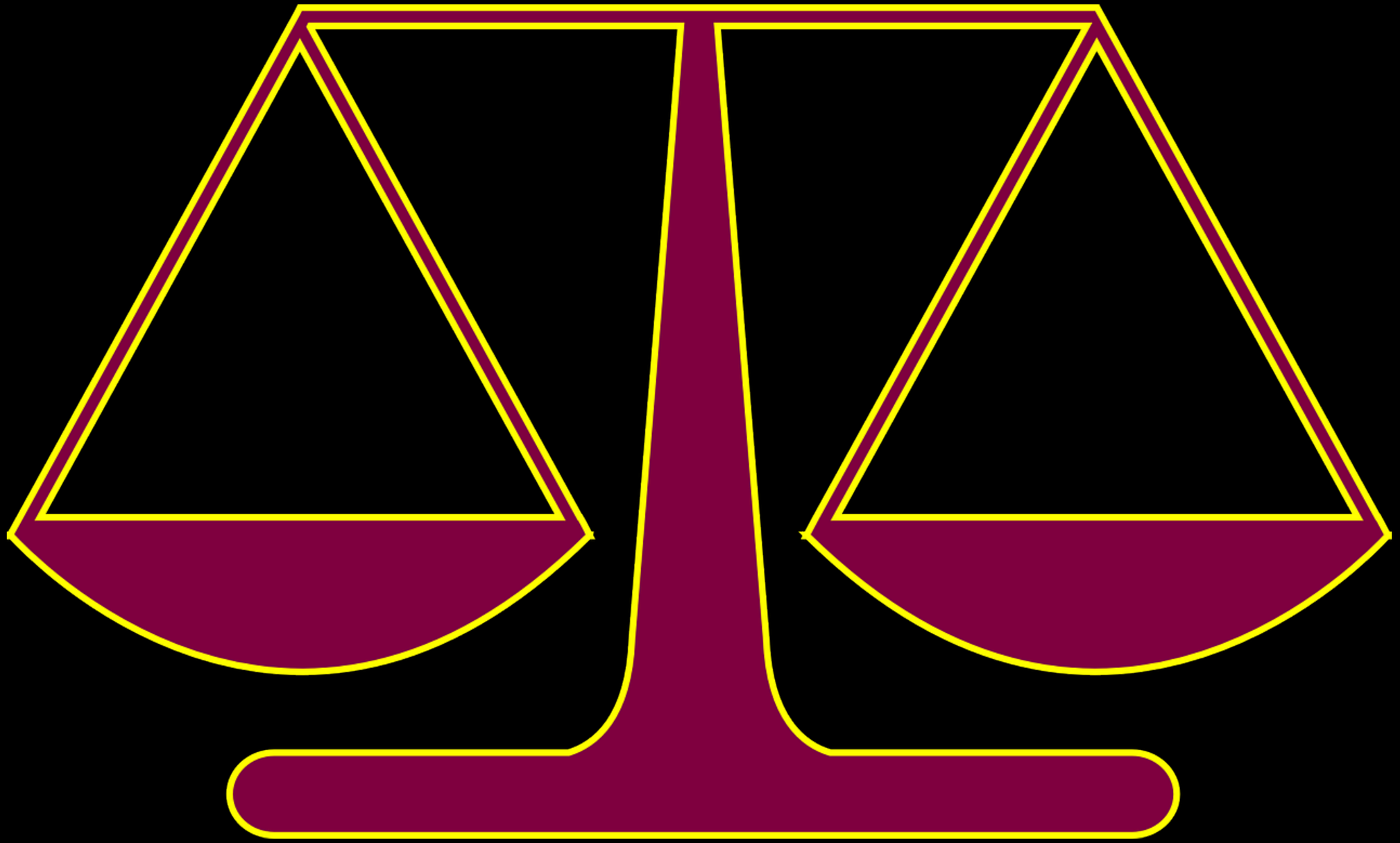
Sie müssen JHipster nicht in Produktion nutzen!

JHipster-Projekt neben Produktions-Projekt "zum Abgucken"

- Wie integriert man das Build-System für SPA in Java?
- Welche Test-Frameworks für JavaScript sind gerade aktuell?
- Wann kann ich auf die neue Java- oder Spring-Version wechseln?

Technologie-Vergleich als Entscheidungsgrundlage

- Je zwei JHipster-Projekten, die sich nur in einer Konfiguration unterscheiden
- React vs. Angular, MySQL vs. MongoDB, Maven vs. Gradle…



Wann lohnt sich JHipster?

Sie oder Ihr Team kämpfen mit Angular, React, NoSQL oder Microservices

Blitzstart für Ihr Projekt verschafft Ihnen Zeit, erlaubt "Training on the job"

Sie wissen nicht, wie DevOps, Container und die Cloud Java beeinflussen

Einheitliches Build-System, Tests und Unterstützung für Continuous Integration, Docker & Cloud sparen viel Zeit

Sie finden keine guten Online-Trainings für diese neuen Technologien

Produktionsreifer Code nach Best Practices zum Selbst-Studium mit Java-Backend und Java-Tools erleichtert Absolvieren anderer Kurse

Wann passt JHipster nicht beim Erstellen einer Applikation?

Front End Platform: Weder React, Angular noch Vue.js werden eingesetzt

Java-Version: Java 8 or Java 11 dürfen nicht eingesetzt werden

Spring oder Spring Boot: Dürfen nicht eingesetzt werden

Applikations-Server: Bibliotheken (wie z.B. Spring oder Hibernate) nicht mit Applikations-Server kompatibel

Open Source Licensing: Lizenz-Richtlinien verbieten bestimmte Lizenzen, wie LGPL (Hibernate) oder Eclipse Public License (Logback)

Wann passt JHipster nicht beim Lernen?

Technology: JHipster unterstützt nicht gewünscht Technologie (wie z.B. Java EE oder Scala)

Lern-Stil: Lernende können oder wollen nicht das erforderliche Selbst-Studium des JHipster-Codes leisten

Docker in Azure

Was haben Sie gerade gesehen?

Applikation läuft in der Cloud als Microsoft Azure Application mit einem Docker Compose File

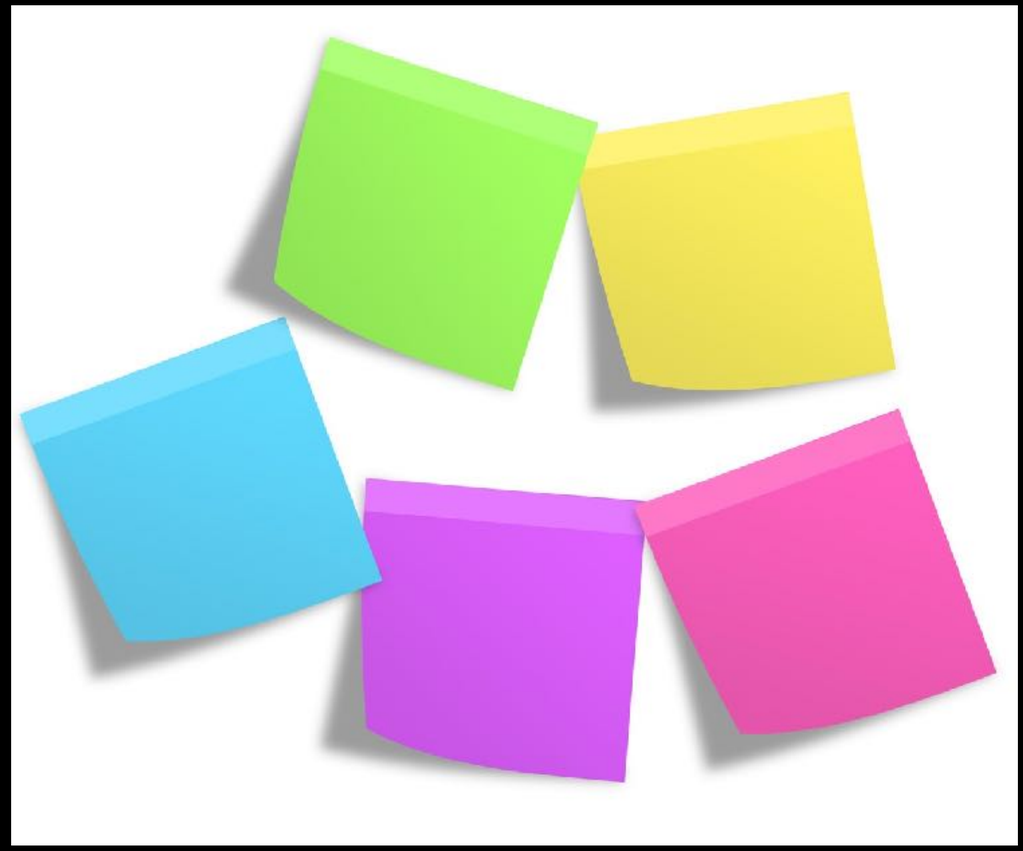
Zusammenfassung

~~Warum JHipster?~~

~~JHipster im Detail~~

~~Demo: "Online-Shop"~~

- ~~Komplett mit JHipster generiert~~
- ~~Continuous Integration, Tests, Docker & Deployment in die Cloud~~



Ihr Chef will

~~Single Page Application (SPA) mit JavaScript~~

~~Open Source~~

~~Container & Cloud~~

~~Agilität & DevOps~~

~~Major Release alle zwei Wochen~~

JHipster liefert schneller bessere Java-Projekte, weil es beim Programmieren, im Projekt und beim Lernen Zeit spart.

Lust auf mehr JHipster?

bpf.li/jfs19

Präsentation

Source Code

Anmeldung zur Mailing List rund um JHipster & Docker

Danke!

Fragen Sie mich:
ksilz@outlook.com

bpf.li/s