Frontend Architektur für Microservices

Thomas Kruse
trion development GmbH
www.trion.de

Training - Beratung - Entwicklung

Kubernetes	Spring Boot	Angular
Docker	Java	React

Thomas Kruse

- Entwickler, Trainer, Berater
  - @everflux
  - www.trion.de
- Java User Group Münster
- Frontend Freunde Münster

Wo stehen wir …

… und wie kamen wir da hin?

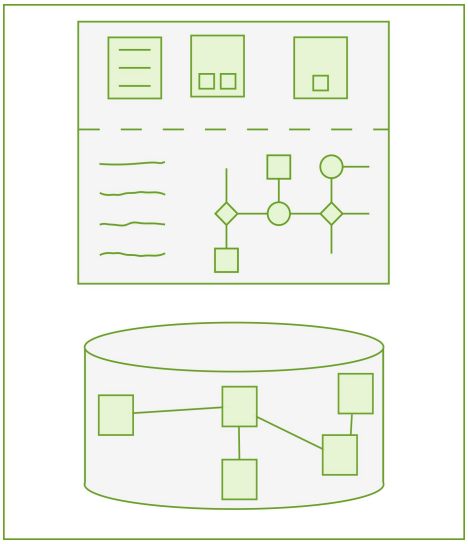
1970 - Zentrale Großrechner

1970 - Nutzer: Experten

```
TRANSACTION DETAIL DEFINITION ---------------------- Applid: APPLHOLT 11:39:07

Internal name ===> CLI-32C               To associate with an entry point name
External name ===> MENUCAPT              Name displayed on user menu
Description   ===> Logon to MENU application (screen capture)
Application   ===> SPCICST        Option  ===> vsr
PassTicket    ===> 0  Name ===>          0=no 1=yes 2=unsigned
Application type  ===> 1                 1=VTAM 2=VIRTEL 3=SERV 4=PAGE 5=LINE
Pseudo-terminals  ===> CLVTA             Prefix of name of partner terminals
Logmode           ===>                   Specify when LOGMODE must be changed
How started       ===> 1                 1=menu 2=sub-menu 3=auto
Security          ===> 1                 0=none 1=basic 2=NTLM 3=TLS 4=HTML
H4W commands ?    ===>                   0=no 1=yes 2=if2VIRTEL 4=auto
Logon message     ===>

TIOA at logon     ===> Signon&/W&*7D4C7D&'114BE9'&U&'114CF9'&P&/Acomplete&/W
MENU&/A
TIOA at logoff    ===>

Initial Scenario  ===>                   Final Scenario     ===>
Input Scenario    ===> CAPTURE           Output Scenario    ===>

P1=Update                      P3=Return                      P12=Server
```

1970 - Architektur: Monolithisch

Linearer Programmfluss



UI (Host Masken)

Geschäftslogik

Zentrale Datenbank

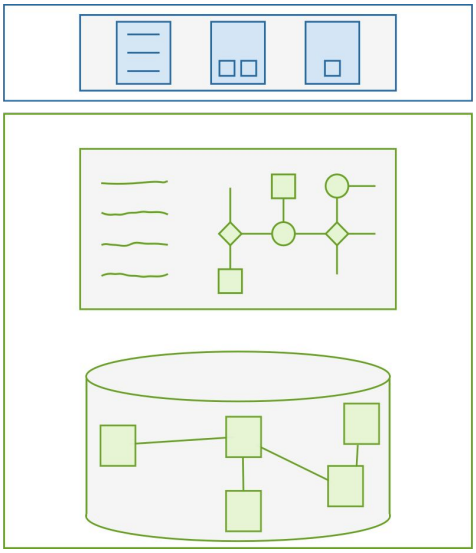
1990 - Schichtenarchitektur

Nutzer haben PCs mit GUI

Anwendungen werden komplexer

Professionalisierung / OO

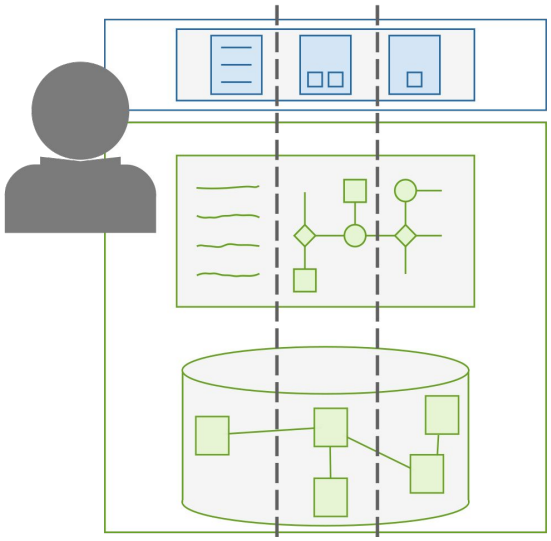
Rich Clients



Frontend

Backend

1990 - Organisation

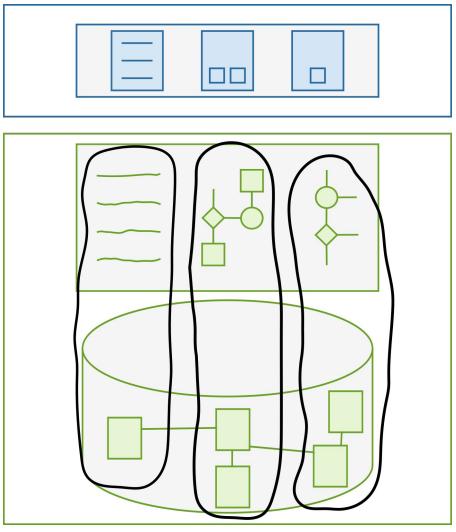


2000 - Modularisierung im Backend

Komplexität steigt

Größenordnungen mehr
Nutzer durch Web

Team Aufteilung:
Framework-/Fachentwickler
Infrastruktur / Betrieb



Anforderungen aus Architektursicht

Komplexität beherschen

Wartbarkeit, Erweiterbarkeit

Analysierbarkeit / Verständlichkeit

Hohe Entwicklungsgeschwindigkeit

Entwickler finden / skalieren können

Gewichtung je nach Kontext

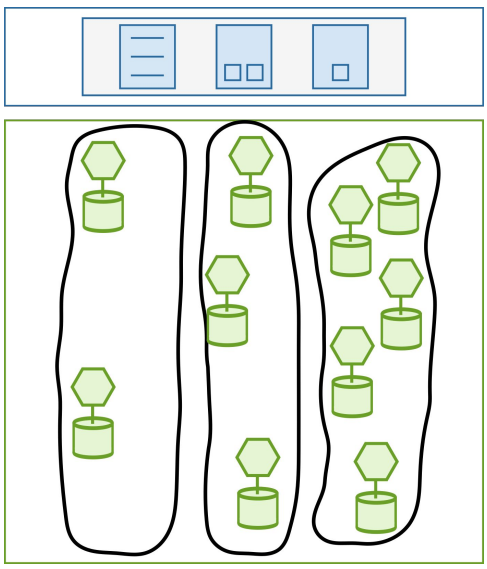
2010 - Microservices

Mobile Nutzer / APIs

Cloud

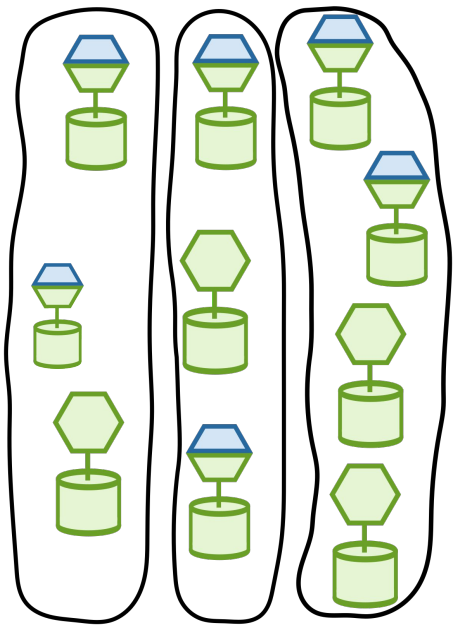
Agile / autonome Teams

Technologische Freiheit



Microservices...

… einfach Microfrontends dazu?



done.

Es gibt da noch den Anwender

Anwender sind **keine** Experten

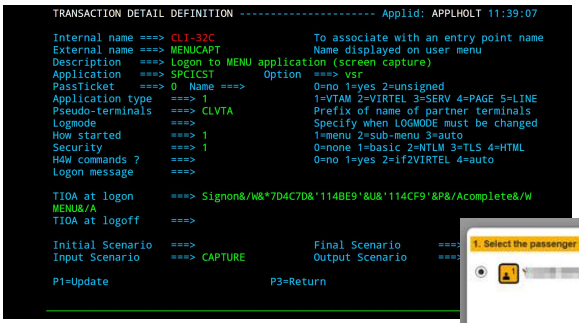
Einheitliche Bedienkonzepte

Hohe Erwartungen an UX

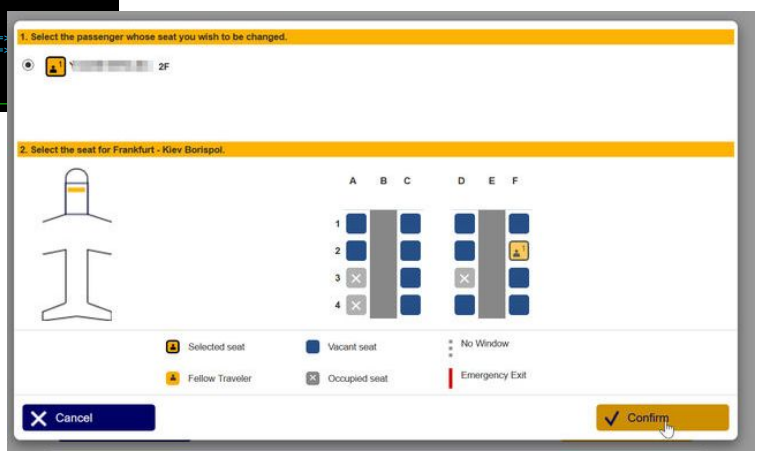
Flüssige Anwendung

Unterstützung statt Formulare

Keine Medienbrüche / Integriertes UI



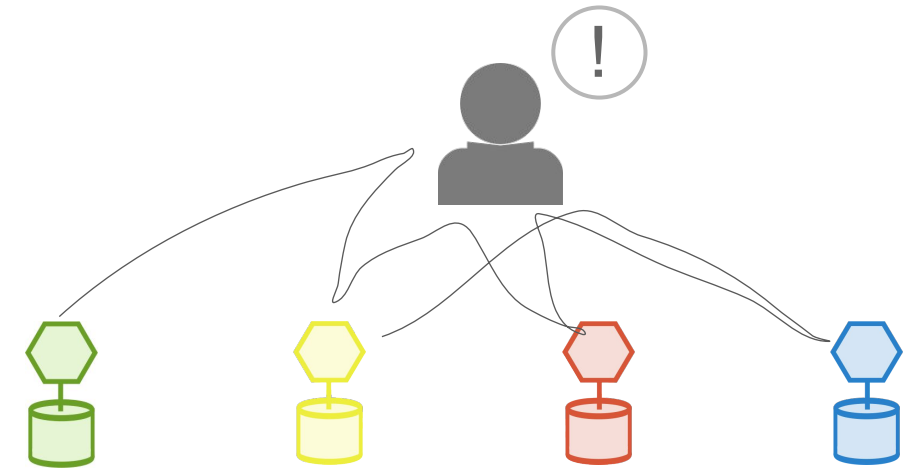
VS.



Beispiel: Reservierung Sitzplatz

Aircraft Parameter | Booking | Leg | Bonus

Beispiel: Reservierung Sitzplatz



Aircraft Parameter | Booking | Leg | Bonus

Beispiel: Reservierung Sitzplatz

UI Microservice

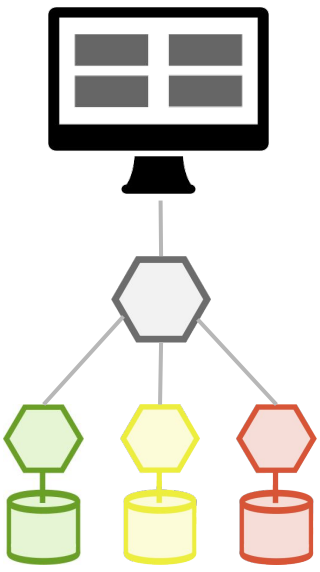
Aircraft Parameter

Booking

Leg

Bonus

Backend for Frontend



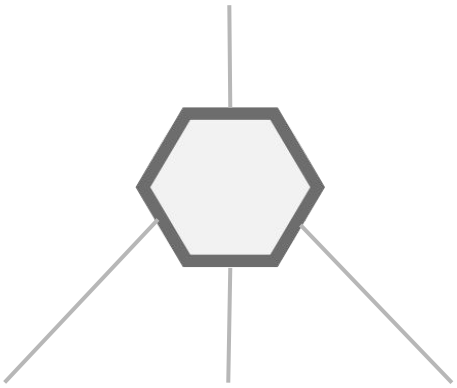
Aufbereitete Ansicht für den Nutzer

Verlagerung von Verantwortlichkeiten in Service

Optimierte API / Format je Frontend

Technische Entkopplung

Backend for Frontend



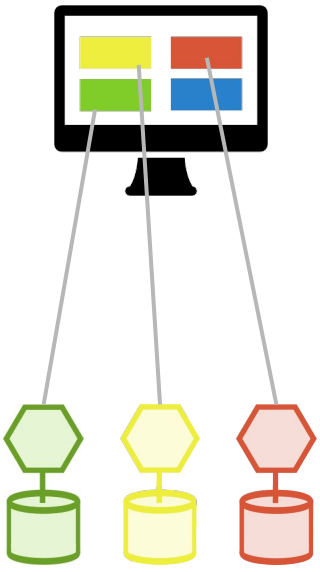
Zu wessen Domäne gehört das BFF?

Anforderungen von Frontend Team

Bereitstellung APIs durch Business Teams

Der nächste Monolith?

Frontend als Integration



HTTP/JSON API mit Browser Anwendung

Option: Cross-Funktionale Teams

Browser Anwendung (SPA)

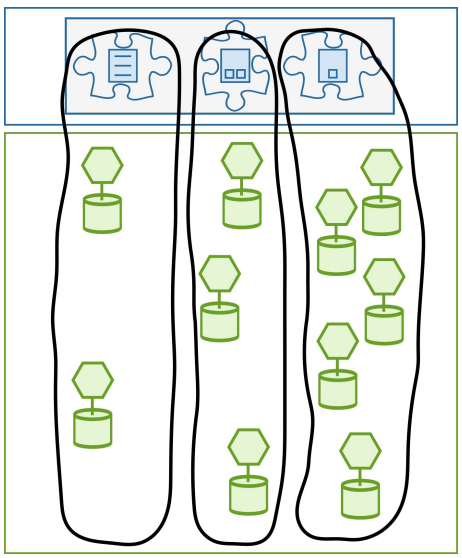
Browser ist Plattform

Produktive Entwicklung mit passendem Framework

Ermöglicht sehr gutes UX

Der Monolith im Browser?

Modularisierung und Wiederverwendung



Umfangreiche Anwendung konzeptionell zerlegen

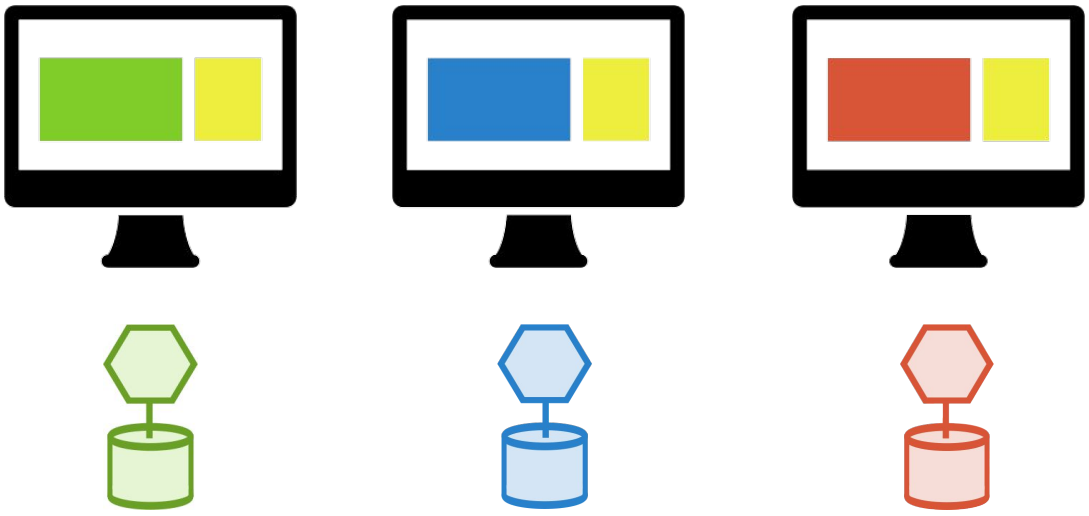
UI **Komponenten**, (JavaScript) **Services**

Module für höherwertige Bausteine

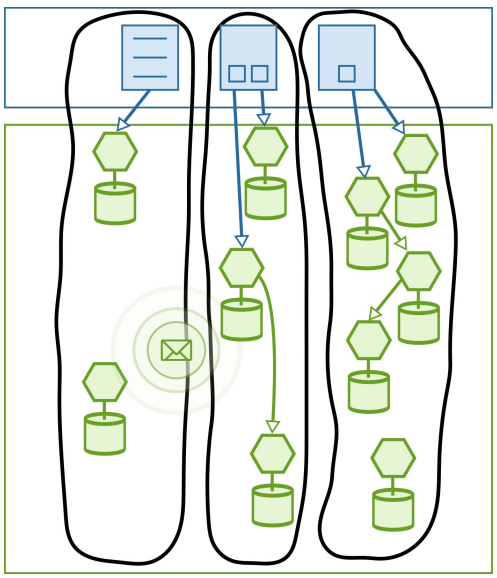
Kopplung an Framework hier bringt Vorteile

Separate Systeme / Anwendungen

Self contained Systems (SCS)



Self contained Systems (SCS)



System durch ein Team verantwortet

Guter Schnitt essentiell (DDD)

Kommunikation bei Systemgrenze: Asynchron

Vorgaben im Unternehmenskontext sinnvoll

Anforderungen des Nutzers

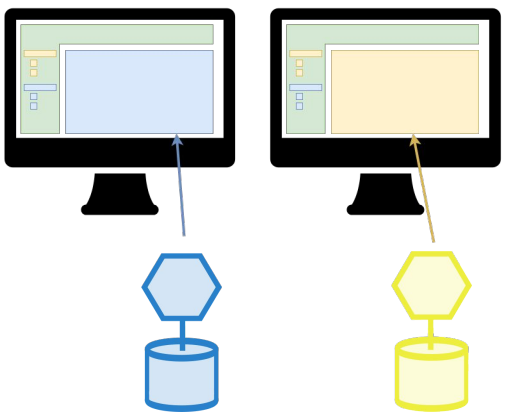
Einstiegspunkte

Übergreifende Navigation

Kontextspezifische Navigation

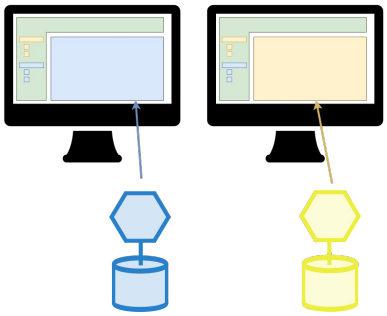
Keine Brüche im Prozess

Integration der Oberflächen: Navigation

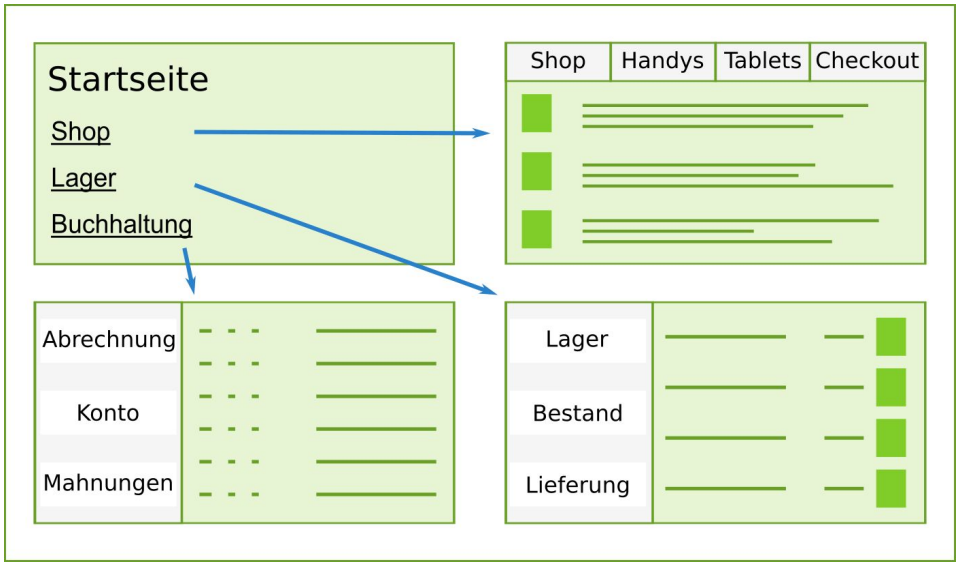


Integration der Oberflächen: Navigation

- Optionen zur Umsetzung der Navigation
  - Startseite mit Einstiegslinks
  - Jedes System liefert Navigation
  - Infrastruktur liefert Navigation



Startseite mit Links

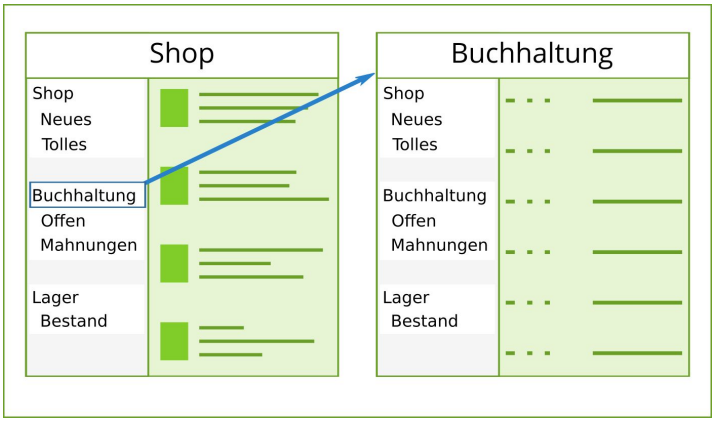


Demo

http://localhost:7000/

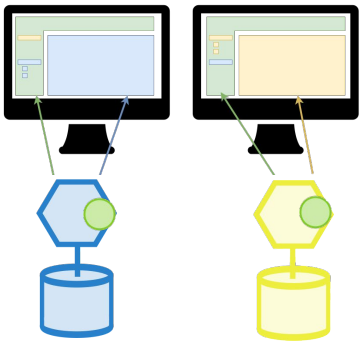
Jedes System liefert Navigation

- Für Benutzer keine Systemgrenze
- Navigationsebenen ggf. limitiert



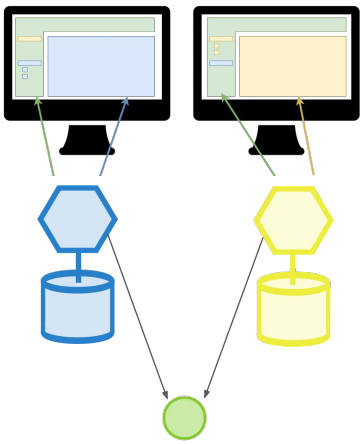
Jedes System liefert Navigation

- Statisch
  - Build Zeit



Jedes System liefert Navigation

- Dynamisch
  - Zur Laufzeit
- Varianten
  - HTML, zentrales Templating (Server side rendered)
  - Daten, lokales Templating



Infrastruktur liefert Navigation

- Frames (deprecated)
- Server Side Includes (SSI), Edge Side Includes (ESI)
- Clientseitiger Abruf zentraler Assets
  - Wird auch als Transklusion bezeichnet

Anforderungen des Nutzers

Einstiegspunkte

Übergreifende Navigation

Kontextspezifische Navigation

Keine Brüche im Prozess

Tiefe Integration

Allgemeine Funktionalität: Newsletter abonnieren

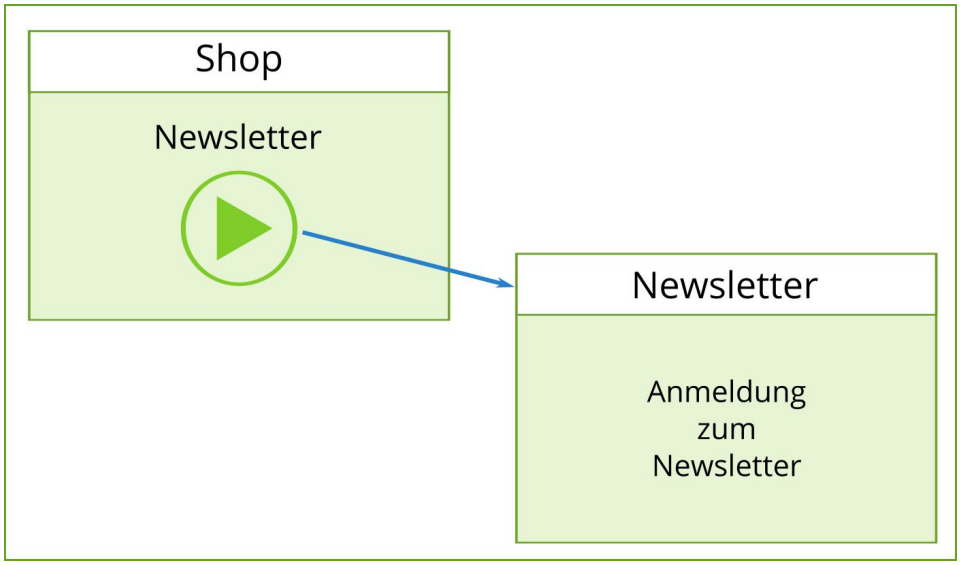
Recherche: Rechnungen eines Kunden

Sprung zu spezifischem UC: Bestellung xy stornieren

Optionen Tiefe Integration

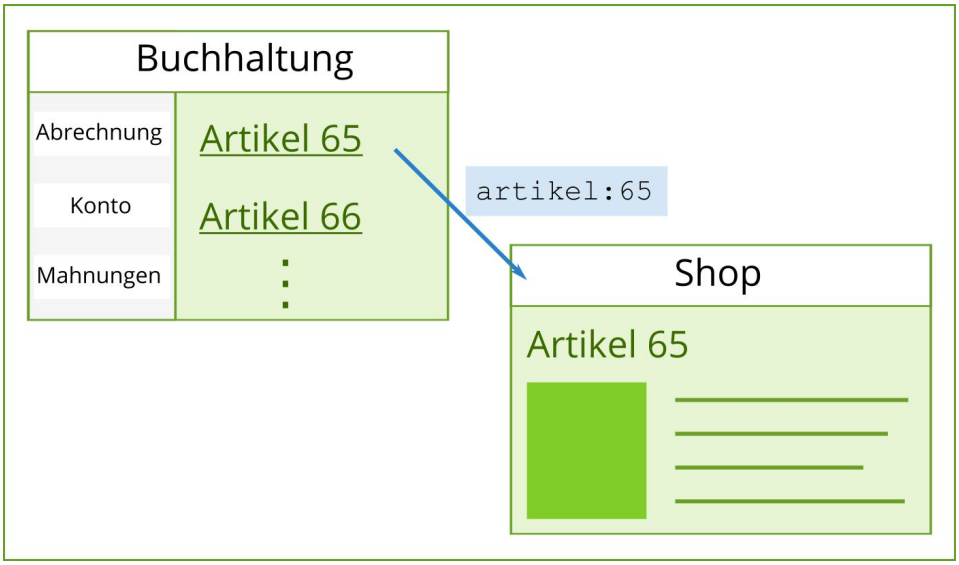
- Link auf Unterbereich von SCS (deep linking)
- Link mit (Rück-)Sprungziel
- Link mit Parameterübergabe
- Link mit Parameterübergabe und (Rück-)Sprungziel
- UI Fragmente (Integration on the Glass)

Link auf Unterbereich

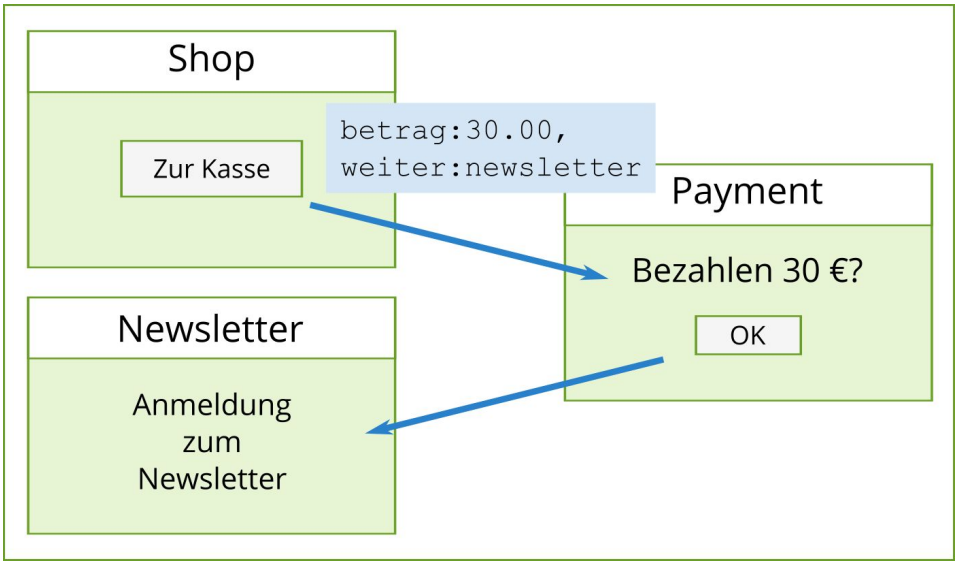


Link mit (Rück-)Sprungziel

Link mit Parameterübergabe



Link mit Parameterübergabe und (Rück-)Sprungziel



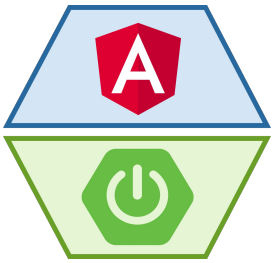
Herausforderungen bei Integration

- Zustand bei Anwendungswechsel
- Security
  - Single-Sign-On
  - Rollen, Berechtigungen
- Grundlage ist guter Schnitt
  - Fragment-Mashup ist schwer handhabbar
  - Führt zu enger Kopplung

Zustand...

… wie funktioniert das mit modernen Anwendungen?

Demo



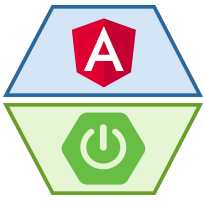
Angular + Spring Boot

Gemeinsame Navigation zur Build Zeit

Reloadfähig, navigierbar

http://localhost:8080/

Angular und Spring Boot



Vorgehen repräsentativ für andere SPA Frameworks

HTML5 Routing verwenden (Server-Side Rewrite Rules)

Angular Modulsystem ermöglicht große, wartbare Anwendungen

Angular Features: Crawlbar (SEO), I18N, Wiederverwendung

Wartbarkeit dank TypeScript

Gemeinsames Deployment als Option - oder Container

Erfahrungen

Angst vor Nutzung von Frameworkfeatures (Kopplung)

Domänengrenzen hängen stark vom Zoom-Level ab

"Warenkorb" und "Artikeldarstellung"
vs.
"Shop" und "Buchhaltung"
vs.
"Shopping" und "Cloud"

Organisationsgrenzen geben Anwendungsdomänen vor (Conway)

Tips

Modellierung mit DDD bietet sich an

Prozesse dabei im Auge behalten, Domänensprünge minimieren

Frameworks bieten nicht nur Nachteile - Vorteile nutzen

Wartbarkeit hat auch etwas mit Testabdeckung zu tun

Echte Anforderungen von Scheinanforderungen trennen

Supporting Domains ...

… gibt es Vorschläge?

Beispiele

Suche / Auswahl von Geschäftspartner

Normalisierung / Validierung von Parametern

Postkorb / Nachrichten

Statusinformationen

Web Analytics

Andere machen es so - Google Maps

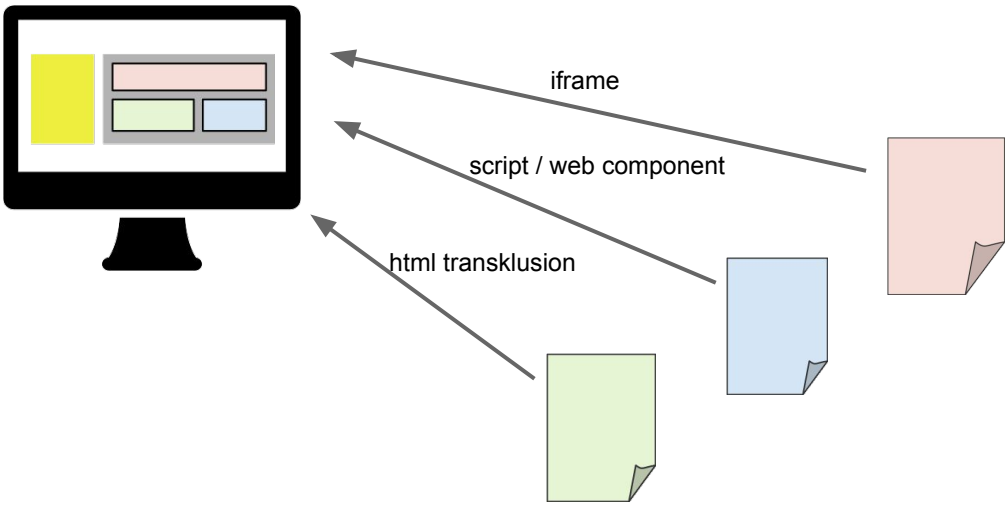
<**script** async defer
src="https://maps.googleapis.com/maps/api/js?**key=**YOUR_API_KEY&**callback=initMap**">
</script>

Andere machen es so - Google Maps

<**script** async defer
src="https://maps.googleapis.com/maps/api/js?**key=**YOUR_API_KEY&**callback=initMap**">
</script>

https://developers.google.com/maps/documentation/javascript/events

Widgets



Widgets

Zugriff auf Backend transparent (inkl. Konfiguration)

Typischer Einsatz: Unidirektionaler Datenfluss nach Initialisierung

Component Varianten: Self-Contained vs. Framework

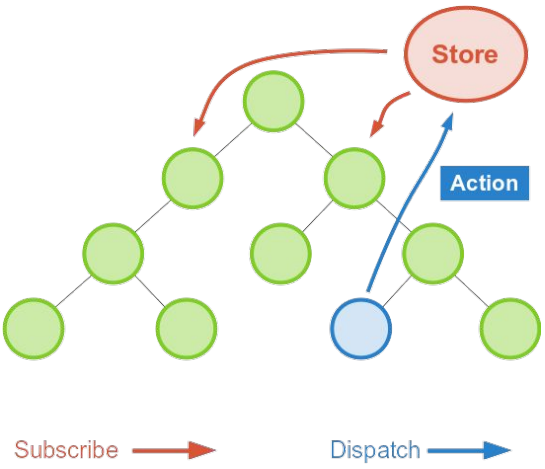
Verwendung Framework-Components

Beispiel: Angular Lazy-Loaded Modules

Vollständige Integration in Anwendung

Partizipiert an Stage-Management

Kopplung an Framework

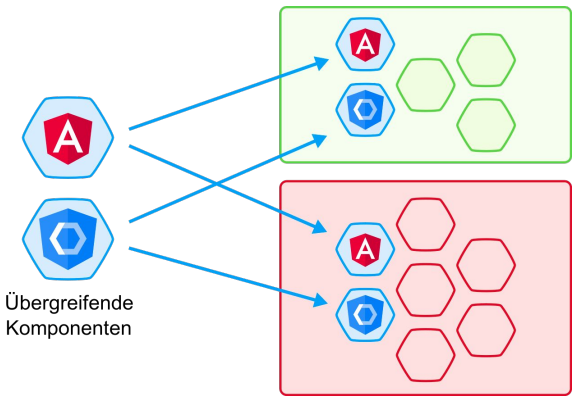


Isolierte Komponenten

Keine Framework Abhängigkeit

Verwendung in beliebigem Kontext

Technologie: Web Components



Web Component Frameworks

API ist low-level, Bibliothek ratsam

Polymer

Angular Elements

…

Demo

http://localhost:7000/news/

Agnostische Komponente ...

… kann man doch orchestrieren!

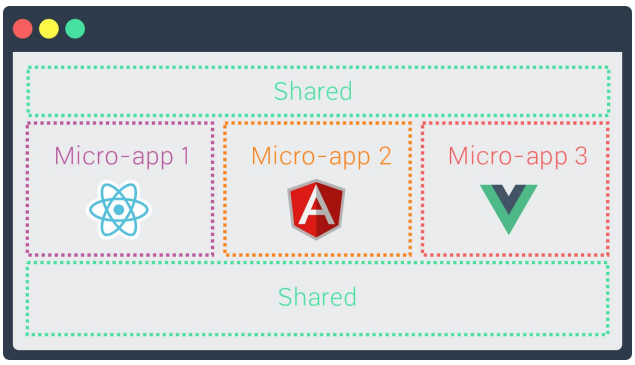
SPA Meta Framework

Technologie: DOM Events, Frames, Meta-Router

Versprechen: Keine Abhängigkeit

Realität: Eigenes Framework, Technologie-Zoo, schwer analysierbar, Performance Impact (~ Portal Server)

Ggf. zur Transition, keine empfehlenswerte Zielarchitektur



Transklusion-Mesh

Versprechen: Keine Abhängigkeit

Realität: Eigenes Framework, API schwer handhabbar, schwer analysierbar, Performance Impact, Herausforderung Security

Ggf. für einzelne View-Only Anreicherungen

Laufzeit Monolith, Gefahr zyklischer Abhängigkeiten!

Danke.

Fragen?

Thomas Kruse

@everflux

TRION

trion.de