Self-Contained Systems Reloaded

Stefan Reuter, Thomas Kruse
trion development GmbH · www.trion.de

Intro

- Stefan Reuter
  - IT-Architekt
  - @stefanreuter

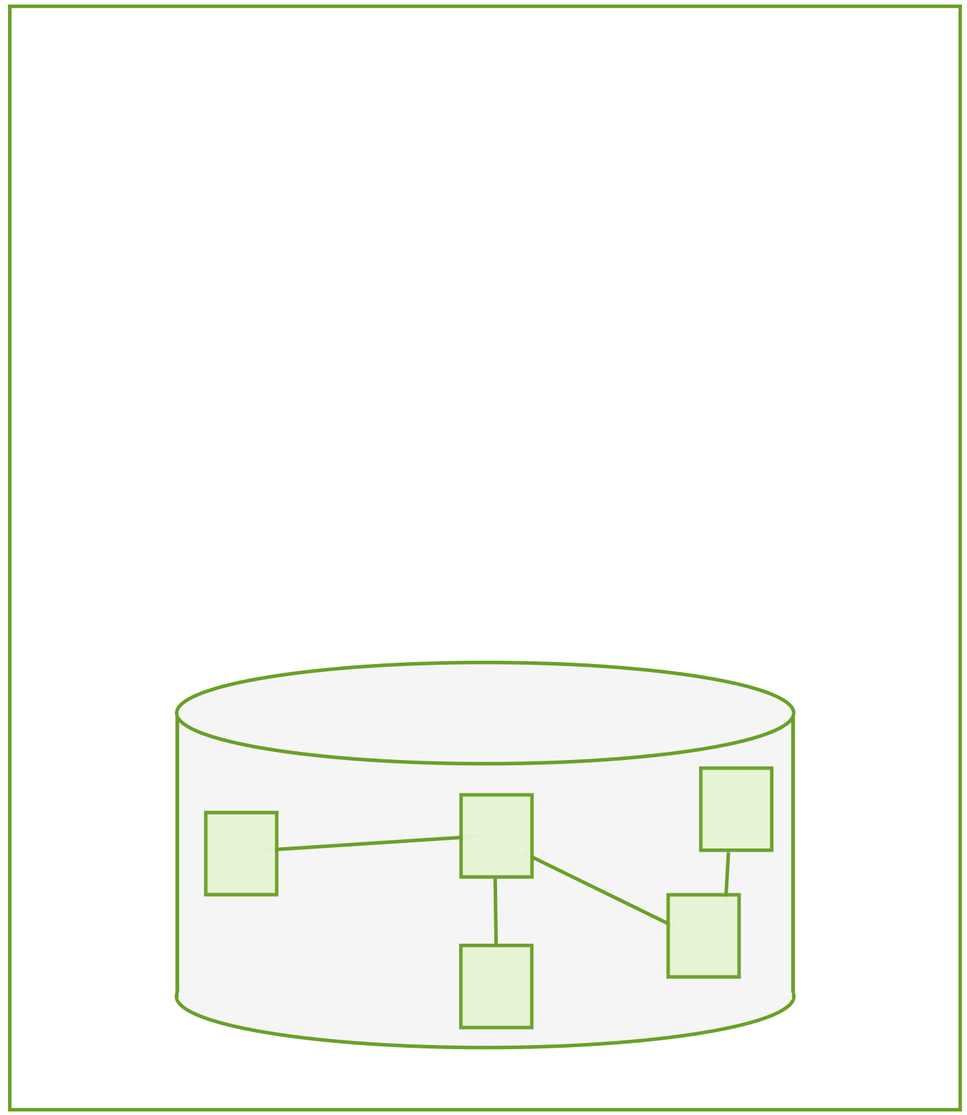
- Thomas Kruse
  - IT-Architekt
  - @everflux
  - JUG Münster, Frontend-Freunde Münster

Agenda

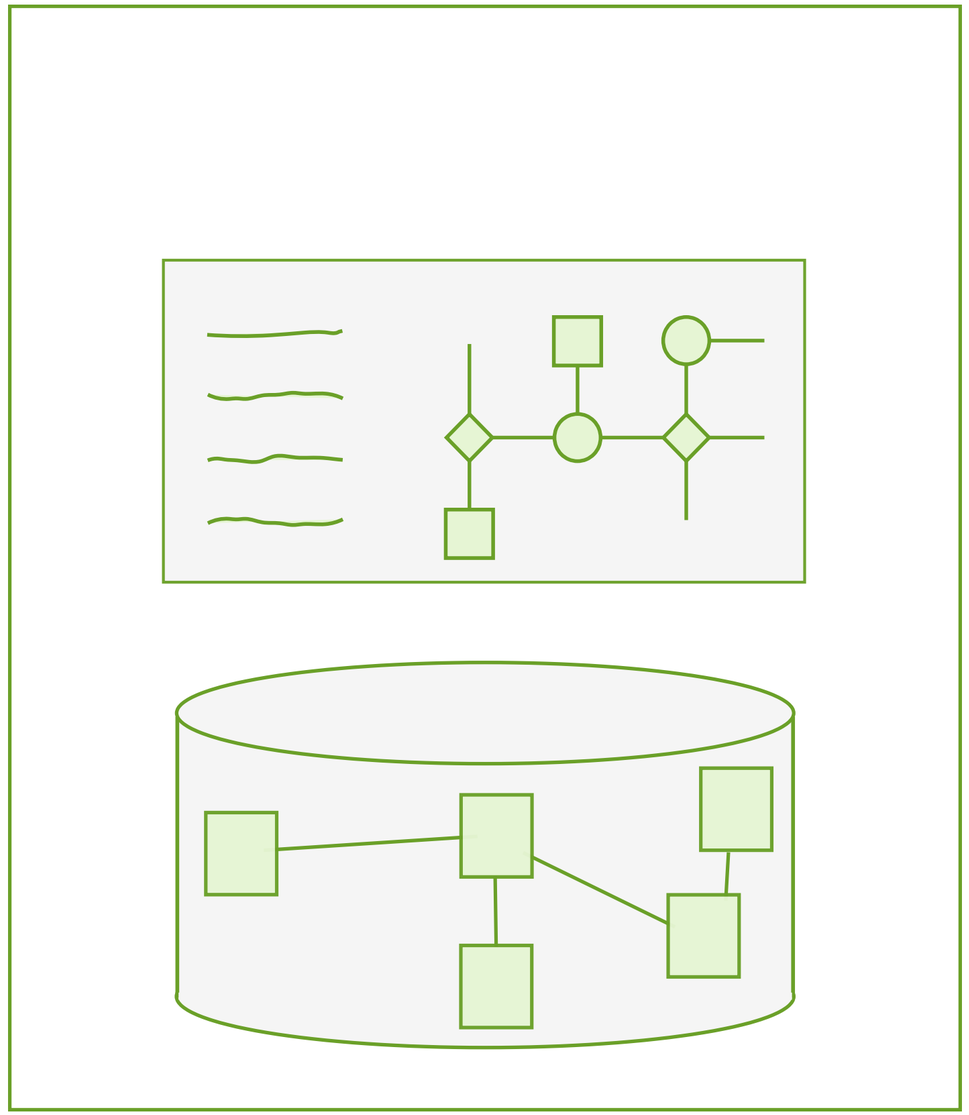
- SCS: Herleitung und Motivation
- Frontend-Integration?
- Moderne (Frontend-)Technologie für SCS

Monolithische Systeme

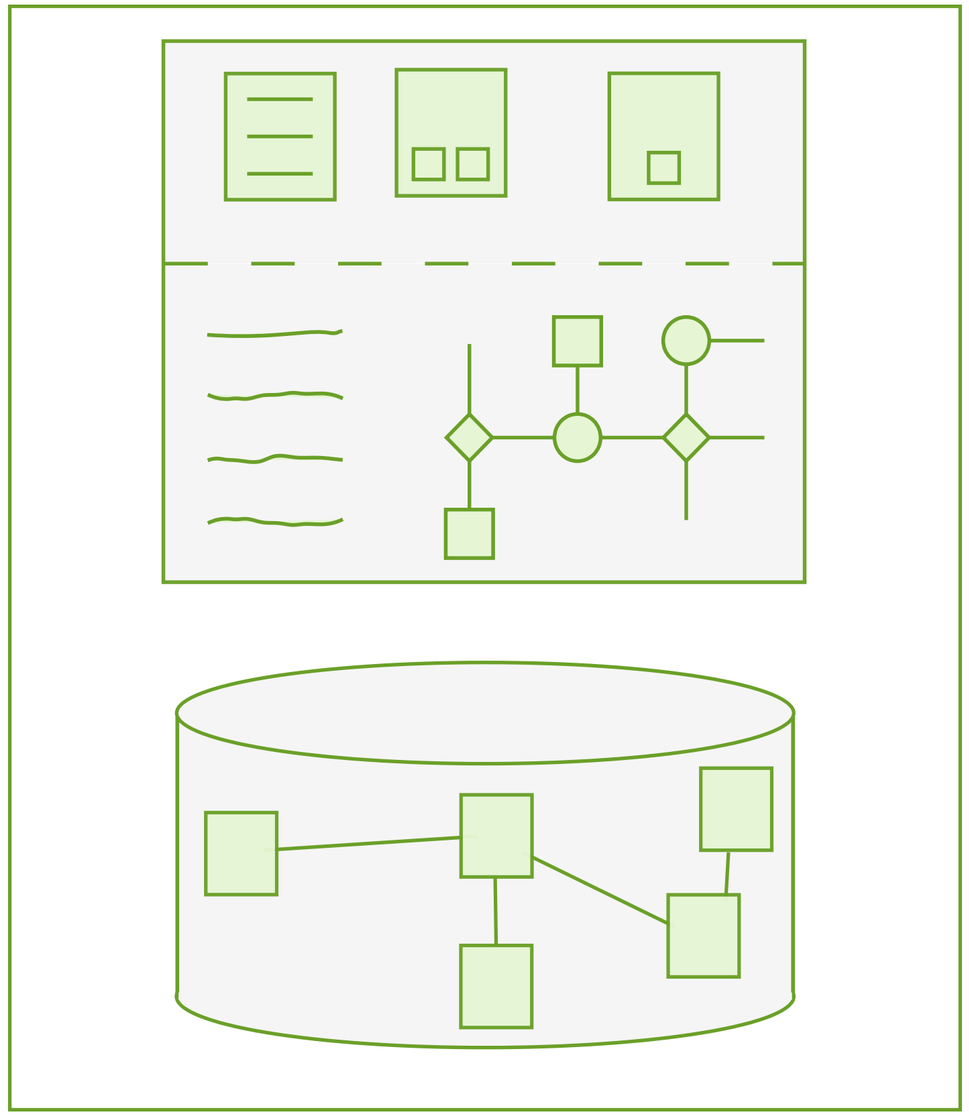
Monolithische Systeme



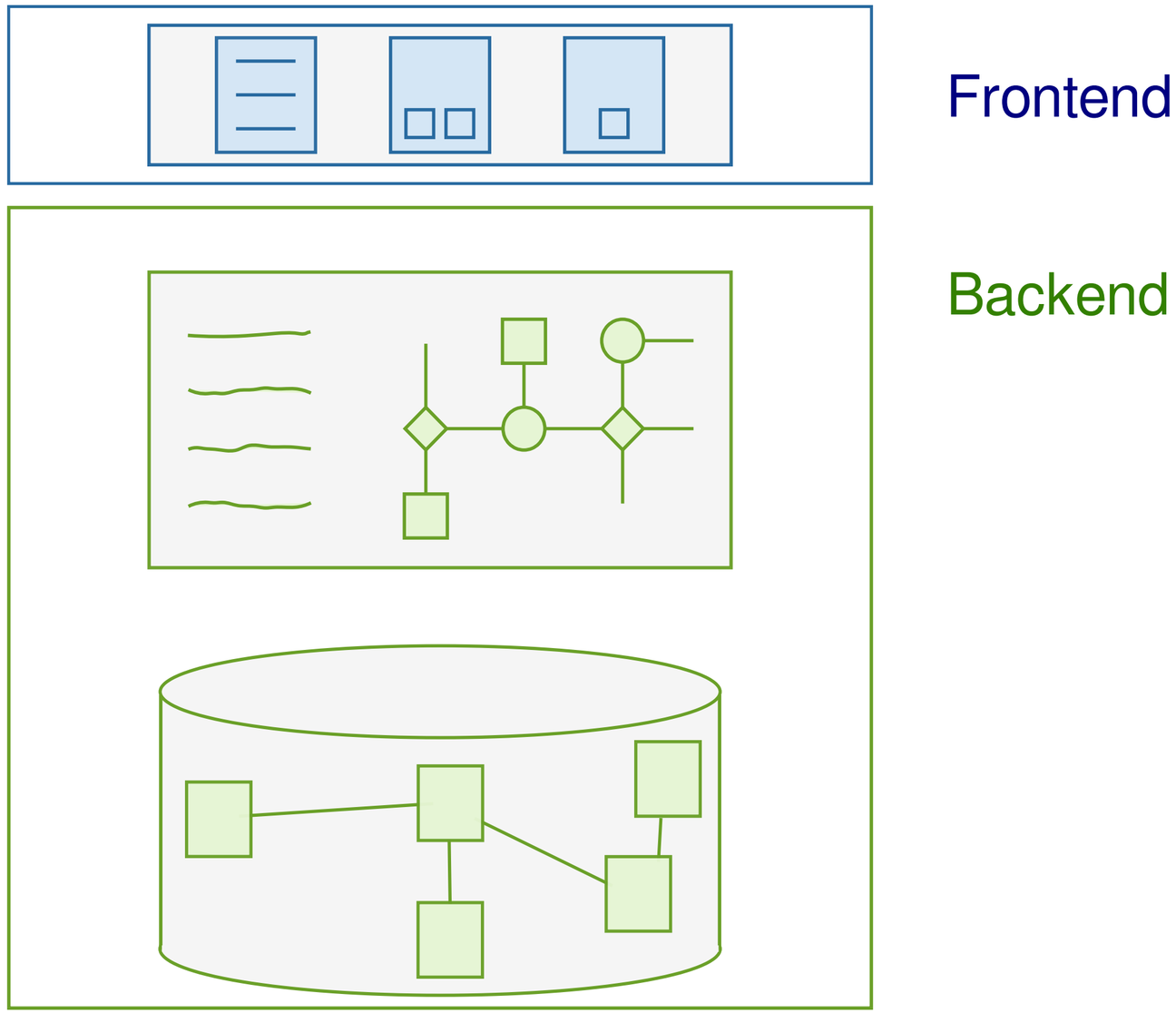
Monolithische Systeme



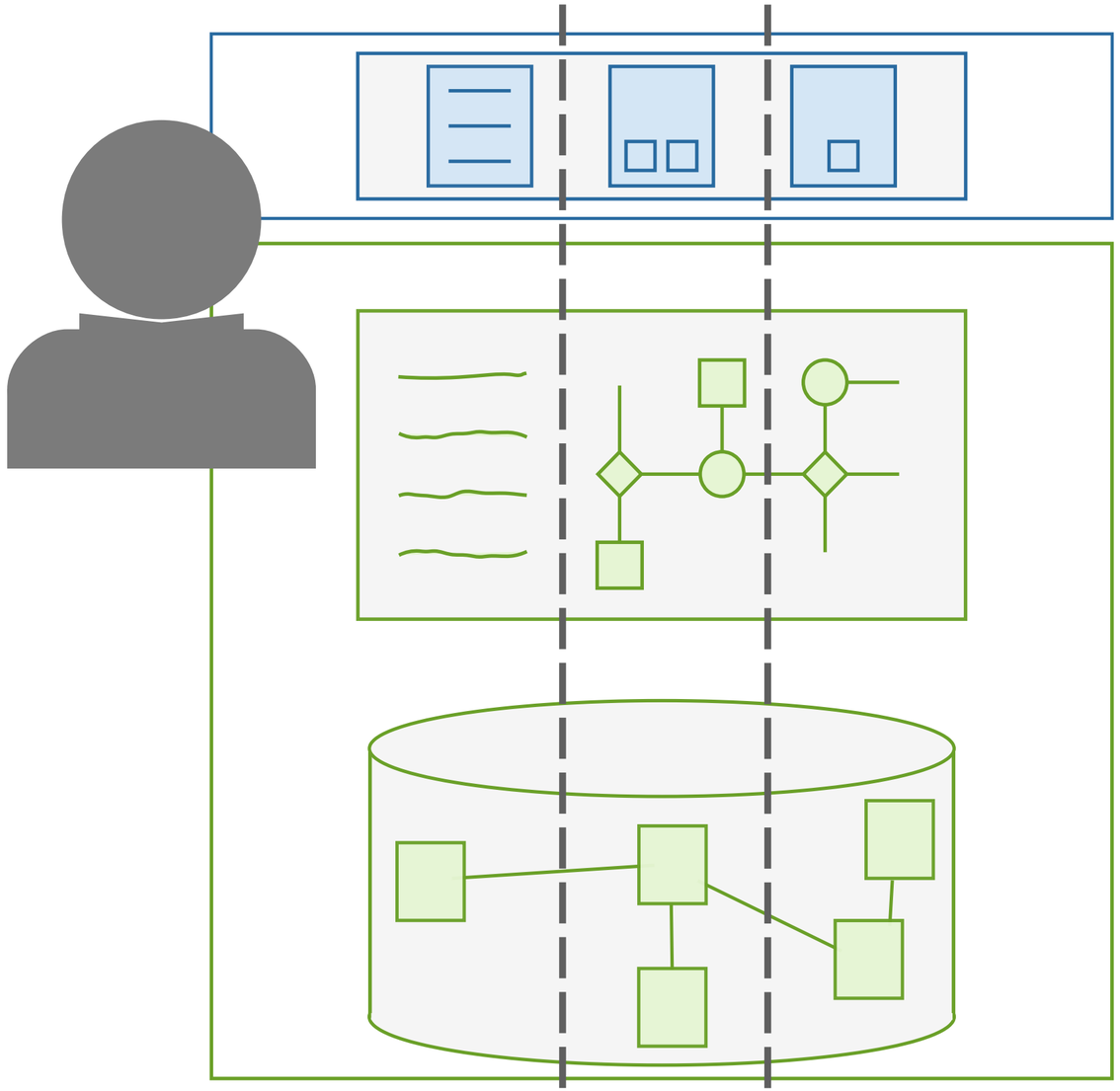
Monolithische Systeme



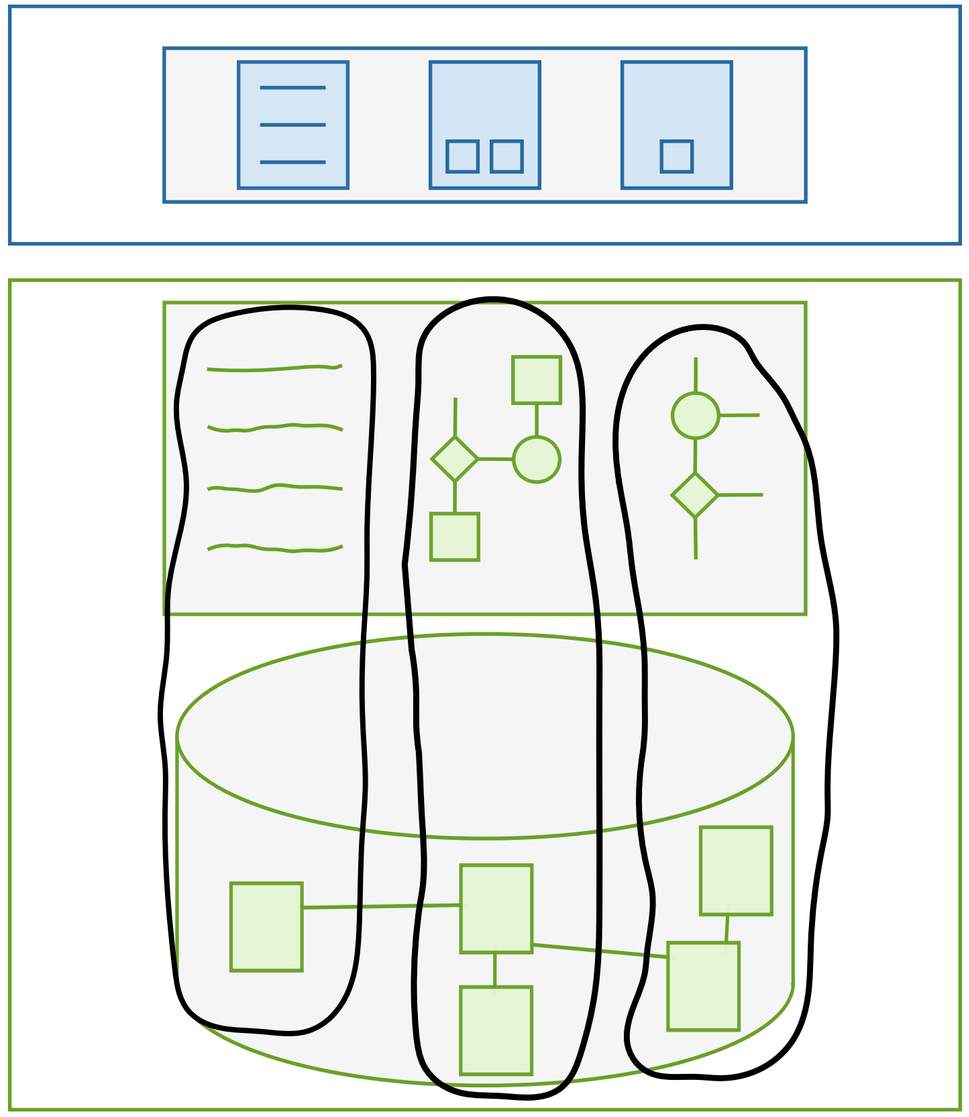
Trennung Frontend/Backend



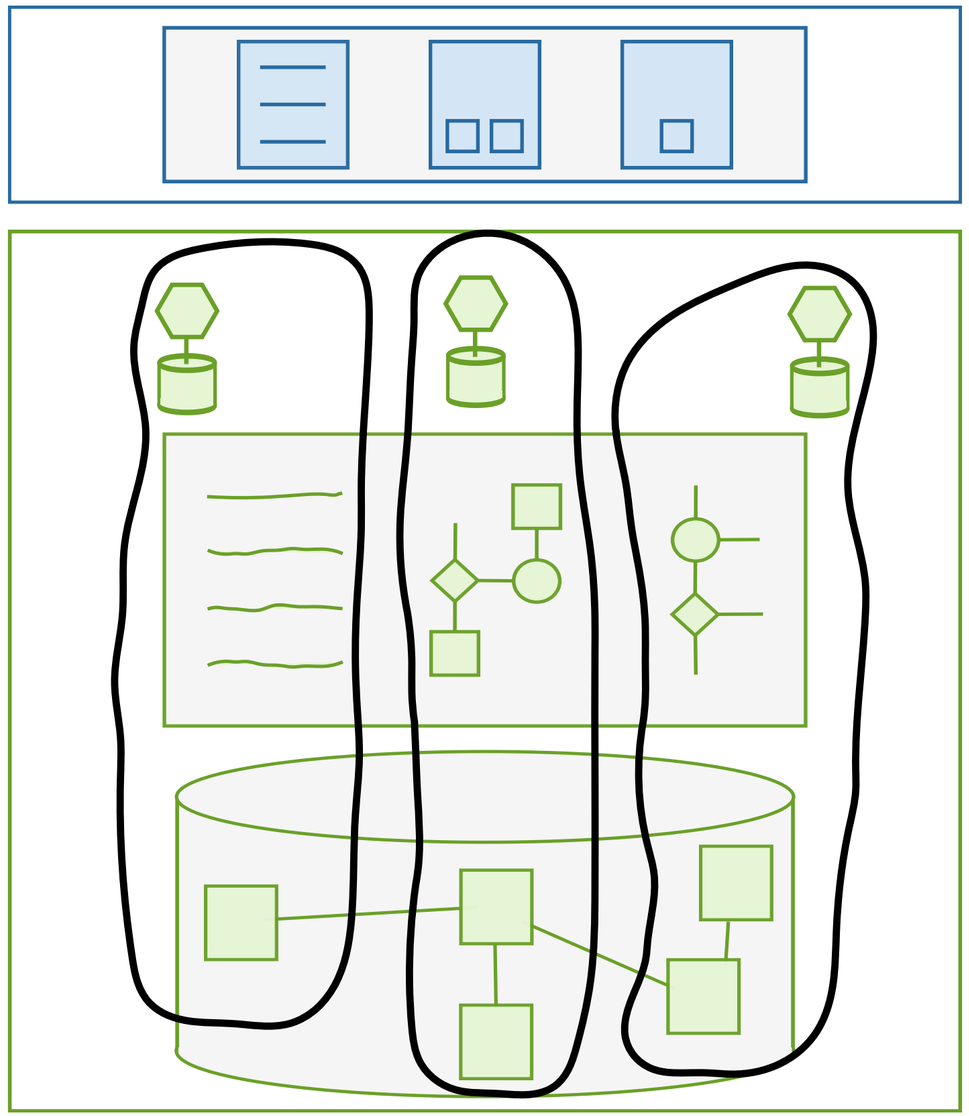
Organisation



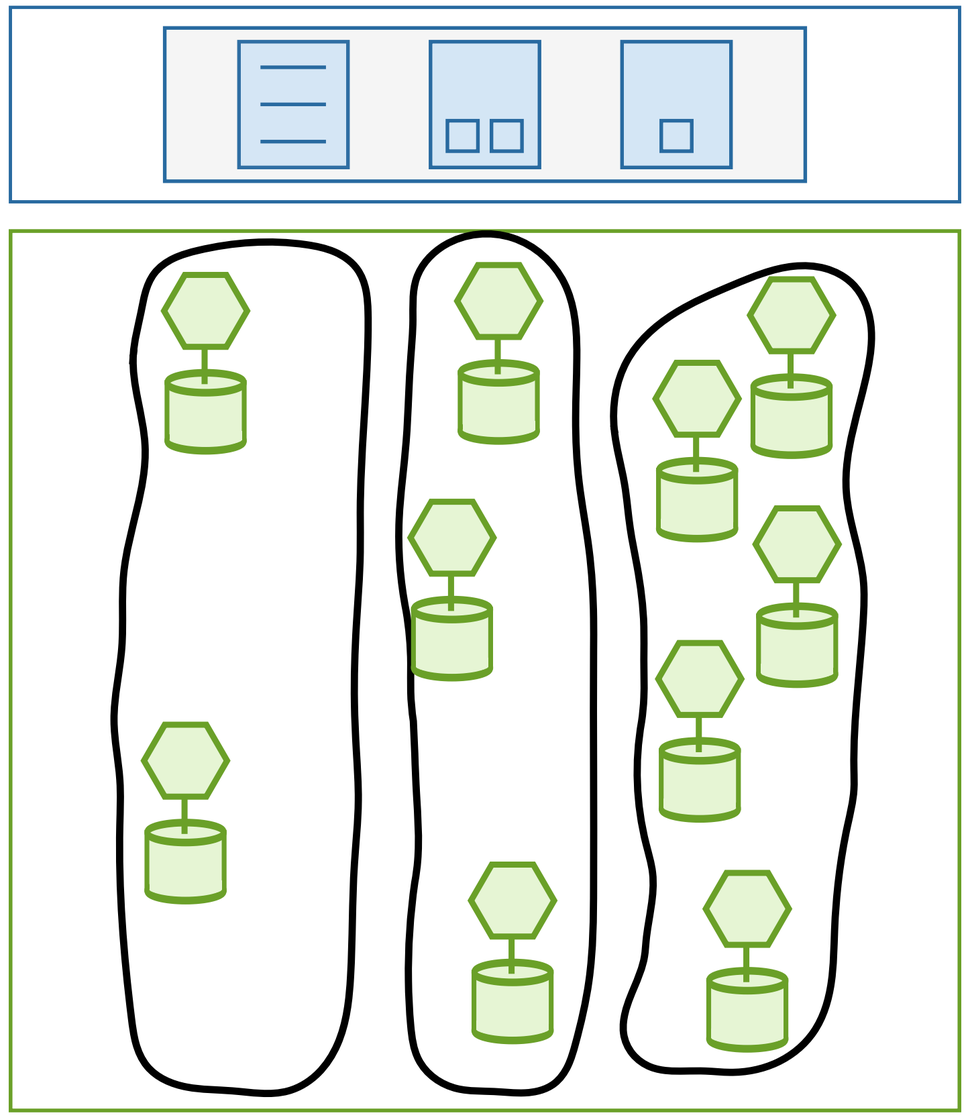
Modularisierung im Backend



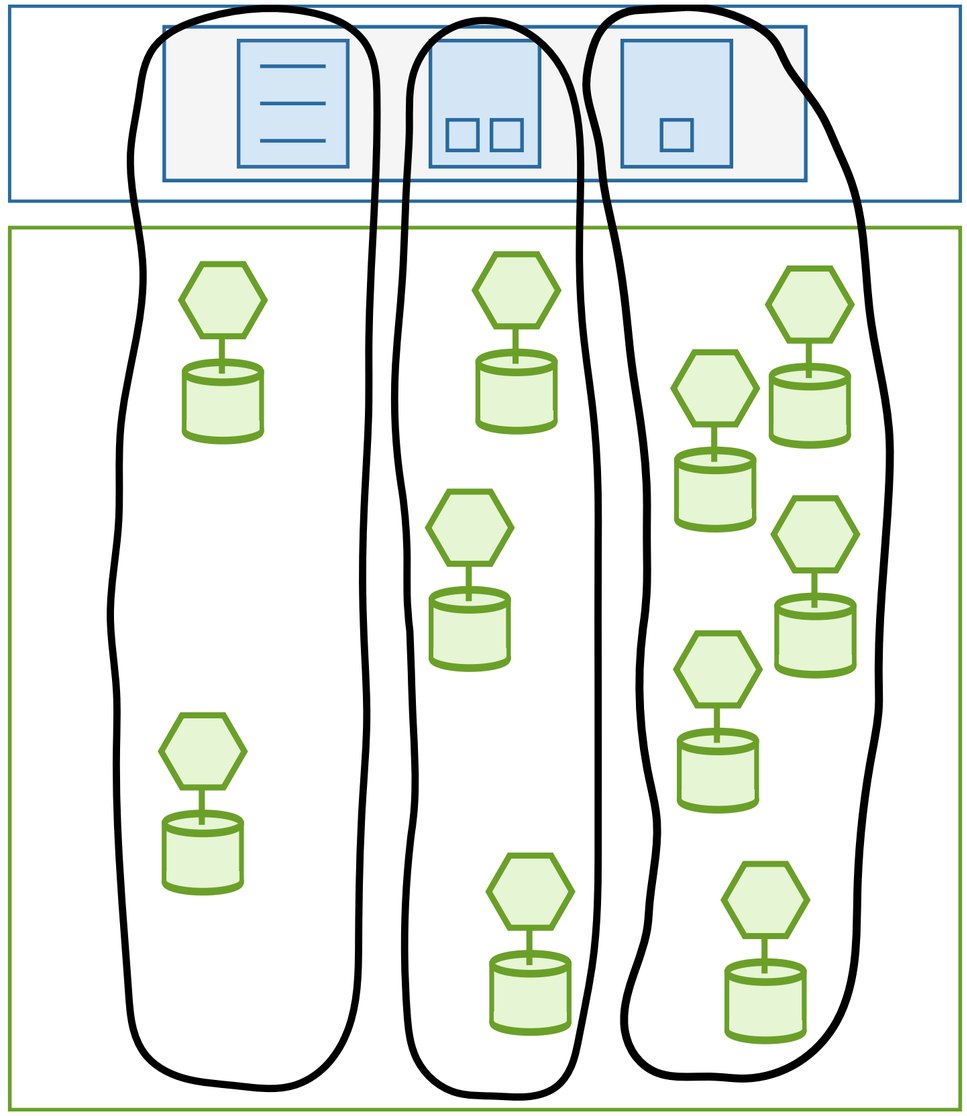
Microservices entstehen



Backend vollständig zerlegt



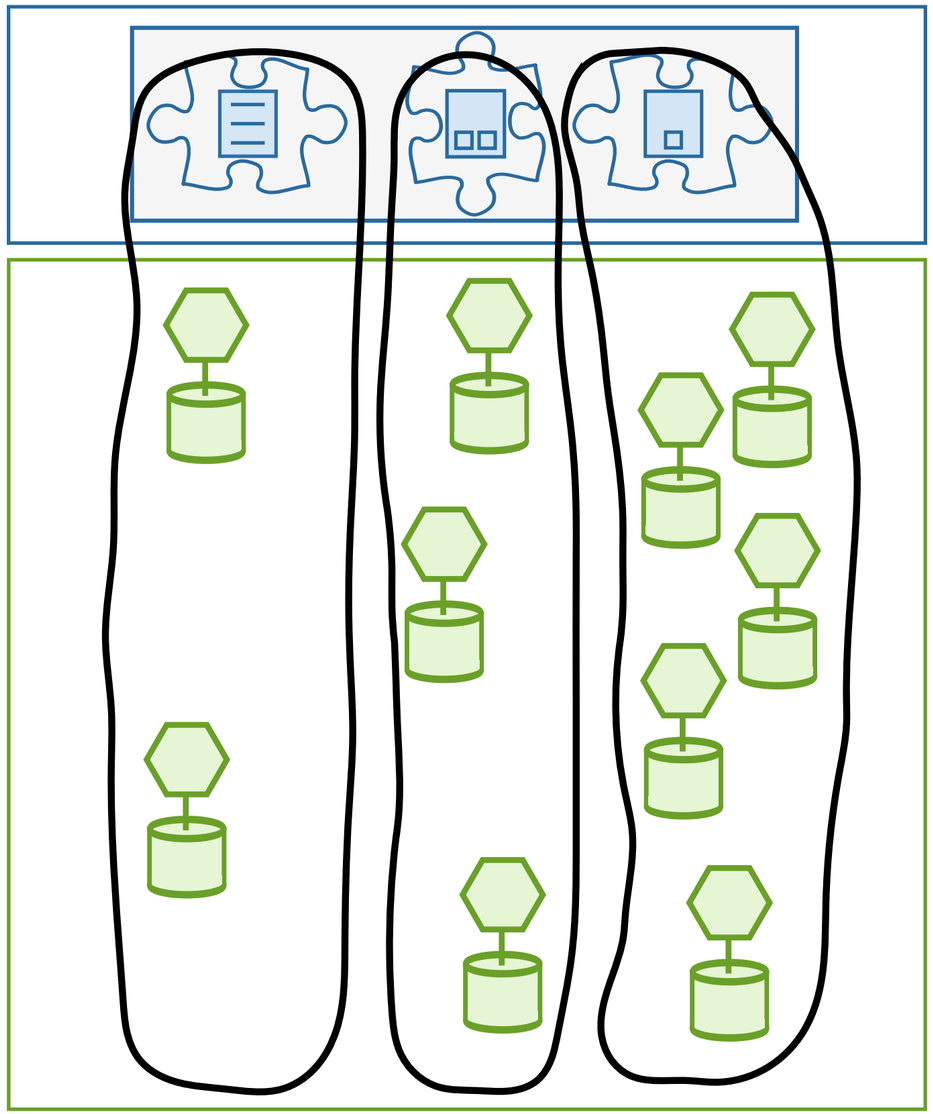
Frontend Modularisierung



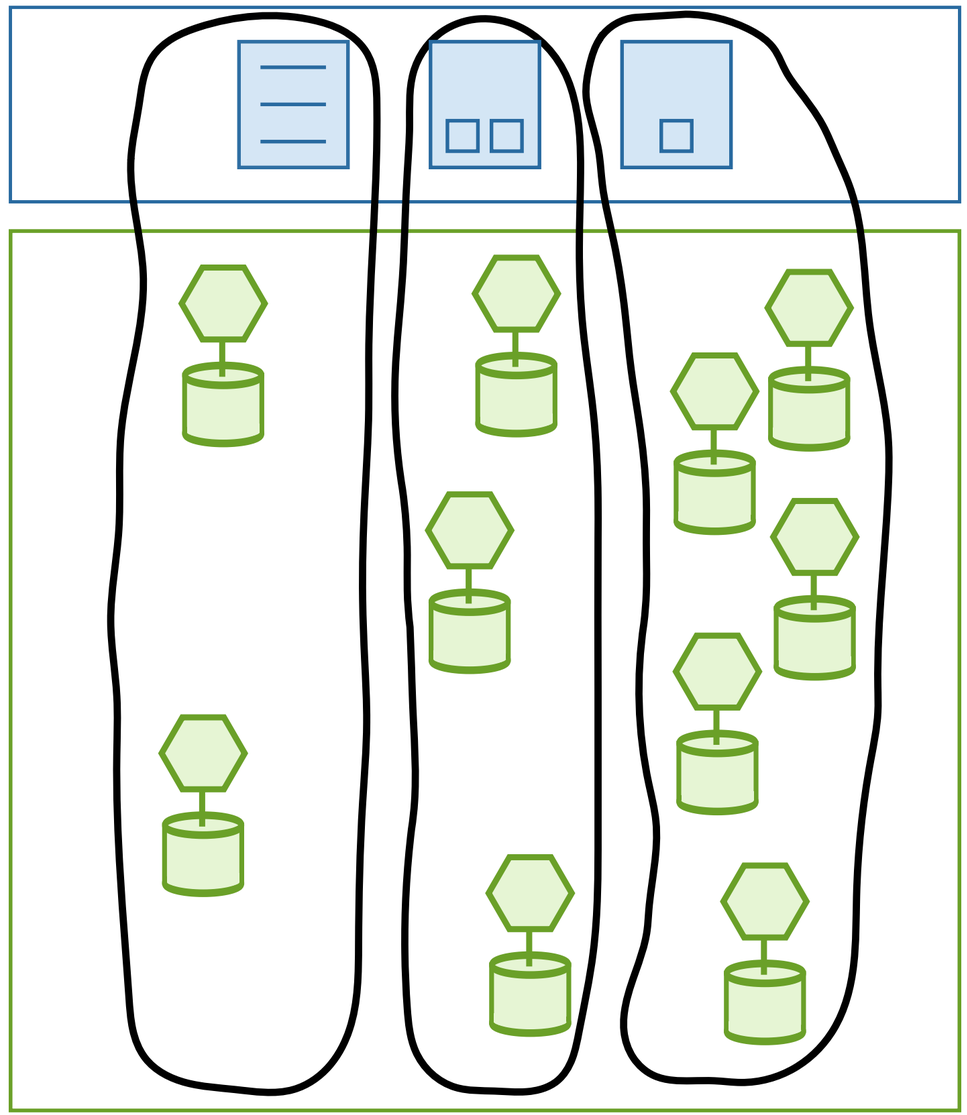
Optionen zur Modularisierung

- Ein modularisiertes Frontend (Tempel-Architektur)
- Ein Frontend je Domäne
- Microservices enthalten Frontend

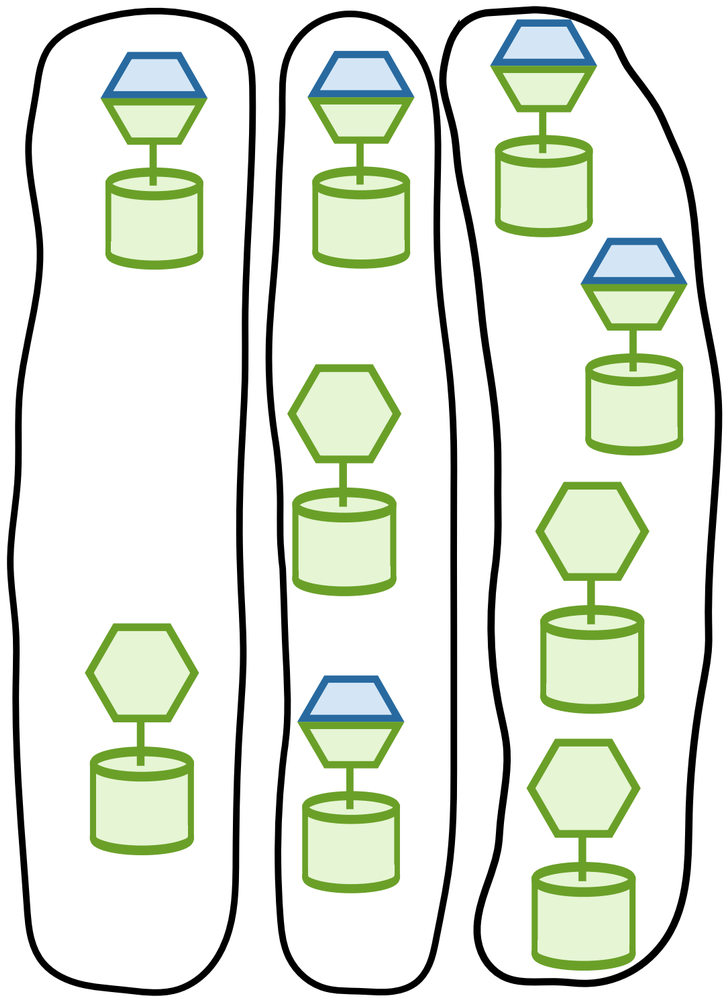
Ein modularisiertes Frontend



Ein Frontend je Domäne



Microservices enthalten Frontend



Self-Contained Systems

1. Jedes SCS ist eine unabhängige Webanwendung.
2. Jedes SCS wird durch ein Team verantwortet.
3. Kommunikation mit anderen Systemen soll - wenn möglich - asynchron erfolgen.
4. Ein SCS kann optional eine Service-API anbieten.
5. Jedes SCS muss Daten und Logik enthalten.
6. Ein SCS soll seine Funktionalität mit eigener UI nutzbar machen.
7. Um enge Kopplung zu vermeiden, darf ein SCS keinen Geschäftscode mit anderen SCS teilen.
8. Um SCS durch verringerte Kopplung robuster zu machen, soll gemeinsame Infrastruktur minimiert werden.

Einordnung Varianten

- Ein modularisiertes Frontend (Tempel-Architektur)
  - Kein SCS, weil keine unabhängigen Webanwendungen
- Ein Frontend je Domäne
  - SCS konform, Schnitt der SCS entspricht Domänenschnitt
- Microservices enthalten Frontend
  - SCS konform, Schnitt der SCS entspricht Domänenschnitt

SCS bringt Vorteile, wenn...

- ...konsequente Modularisierung das Ziel ist.
- ...die Organisation nach Domänen möglich und gewünscht ist.
- ...die volle Autonomie und die damit verbundene Verantwortung gelebt wird.
- ...die Verwendung separater Frameworks bzw. Frameworkversionen als nützlich empfunden wird.

Kommunikation bei SCS

Kommunikation über Systemgrenzen

- Kommunikation zwischen mehreren SCS und zwischen SCS und externen Systemen erfolgt grundsätzlich asynchron.
  - Ziel: Jedes SCS soll autonom auskunftsfähig sein.
- Hilfreich: Nutzung von Domain Events aus Domain-Driven Design.
  - Folgeverarbeitung ergibt sich als Konsequenz von Ereignissen.
  - Daten aus Domain Events können lokal persistiert werden.

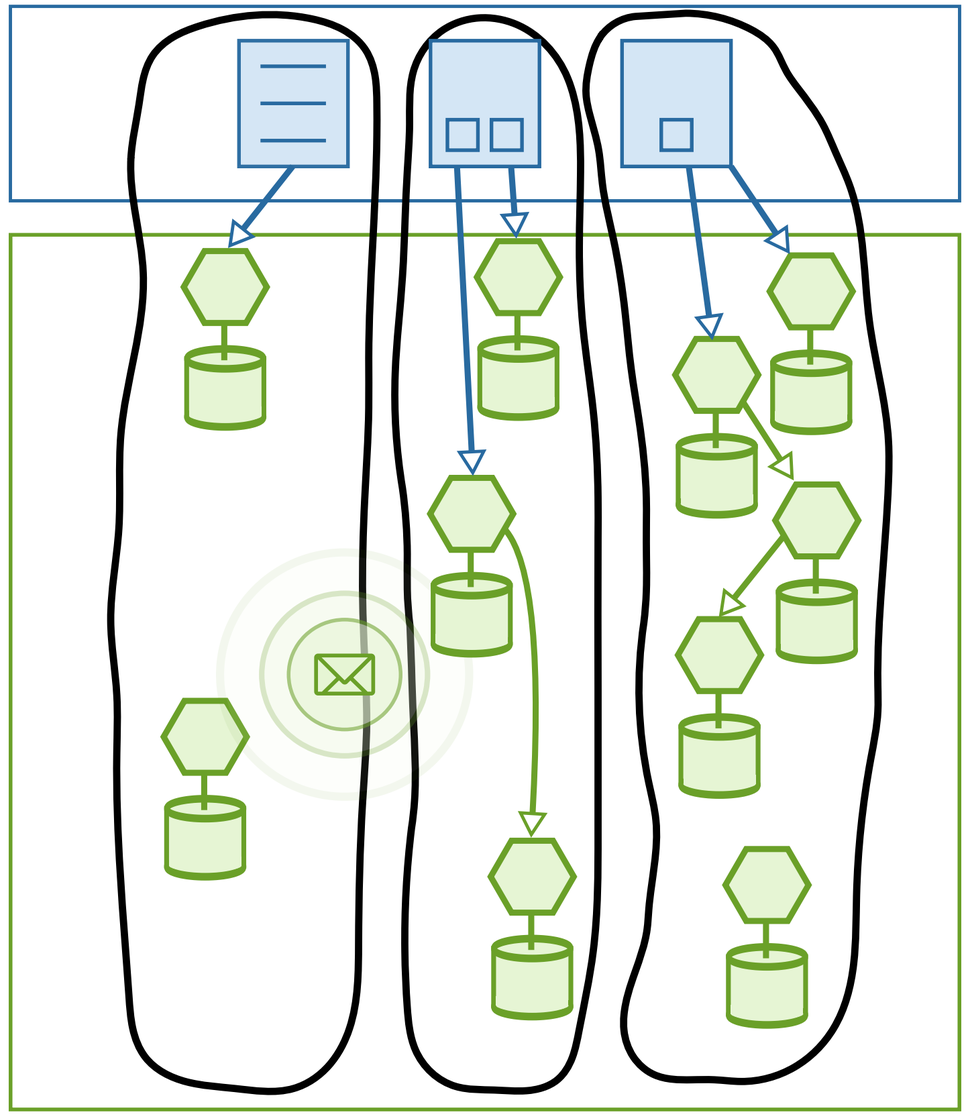
Technologien

- Polling
  - HTTP Feeds, Atom
  - Client kann wieder aufsetzen, Parallelisierung schwierig
- Messaging Publish-Subscribe
  - JMS Topic, AMQP
  - Kafka, Apache Pulsar

Kommunikation innerhalb eines Self-Contained Systems

- Alles erlaubt
  - Services untereinander, Frontend zu Services
- In der Praxis haben sich Vorgaben innerhalb der Organisation bewährt:
  - Technologie-Spektrum begrenzen
  - Lange Aufrufketten vermeiden
  - Konsistenz sicherstellen
  - Robustheit erhalten

SCS Landschaft



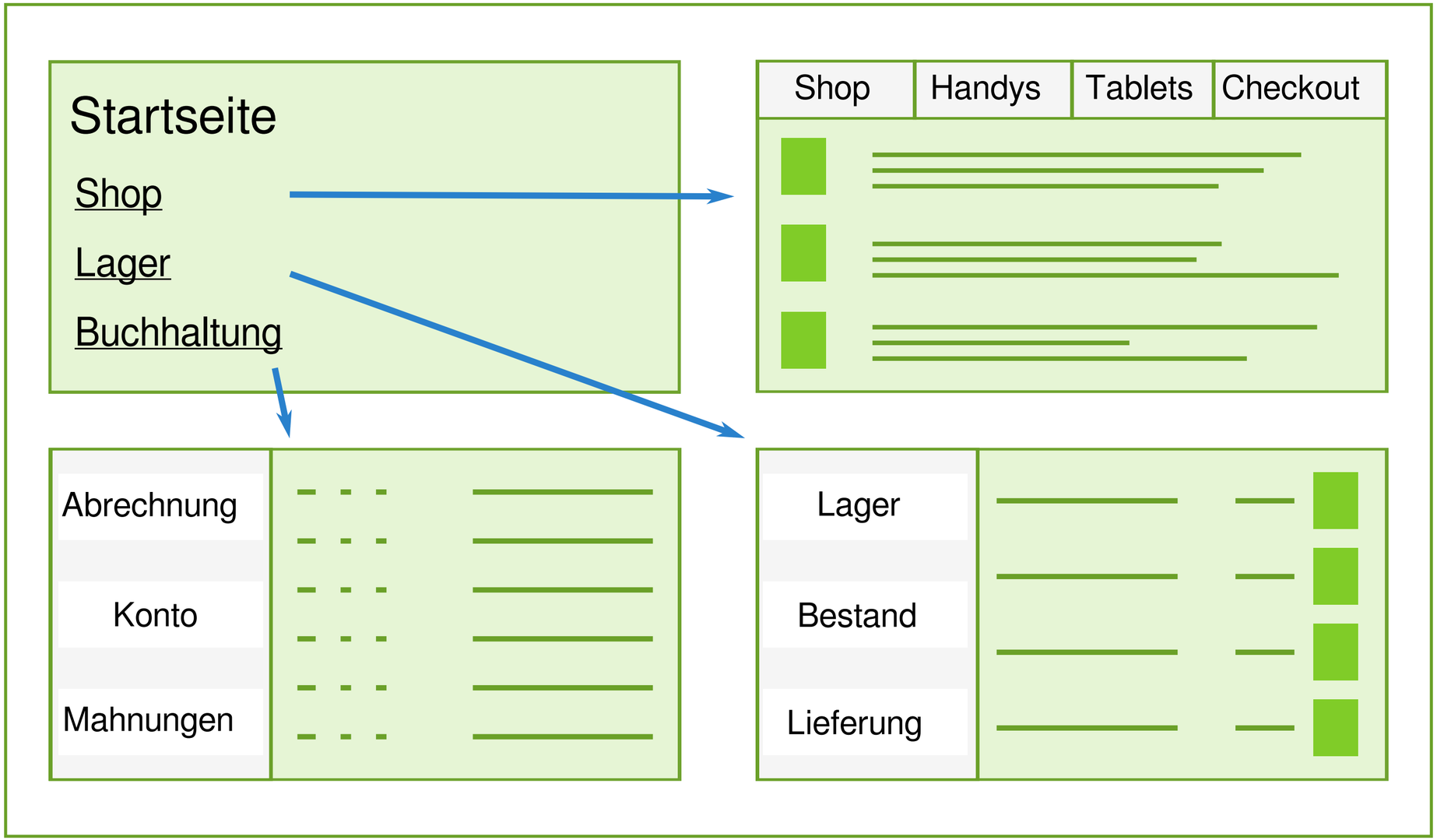
Wünsche der Anwender

- Anforderungen an Navigation
  - Einstiegspunkte
  - Übergreifende Navigation
  - Kontextspezifische Navigation
- Anforderung: Einheitliche Oberfläche (UX und Optik)

Integration der Oberflächen

- Optionen zur Umsetzung der Navigation
  - Startseite mit Links
  - Jedes SCS liefert Navigation
  - Infrastruktur liefert Navigation
- Tiefe Integration
  - Beispiel: Produktvorschläge

Startseite mit Links



Jedes SCS liefert Navigation

- Statisch
  - Build Zeit
- Dynamisch
  - HTML, zentrales Templating
  - Daten, lokales Templating

Infrastruktur liefert Navigation

- Frames (deprecated)
- Server Side Includes (SSI), Edge Side Includes (ESI)
- Clientseitiger Abruf zentraler Assets

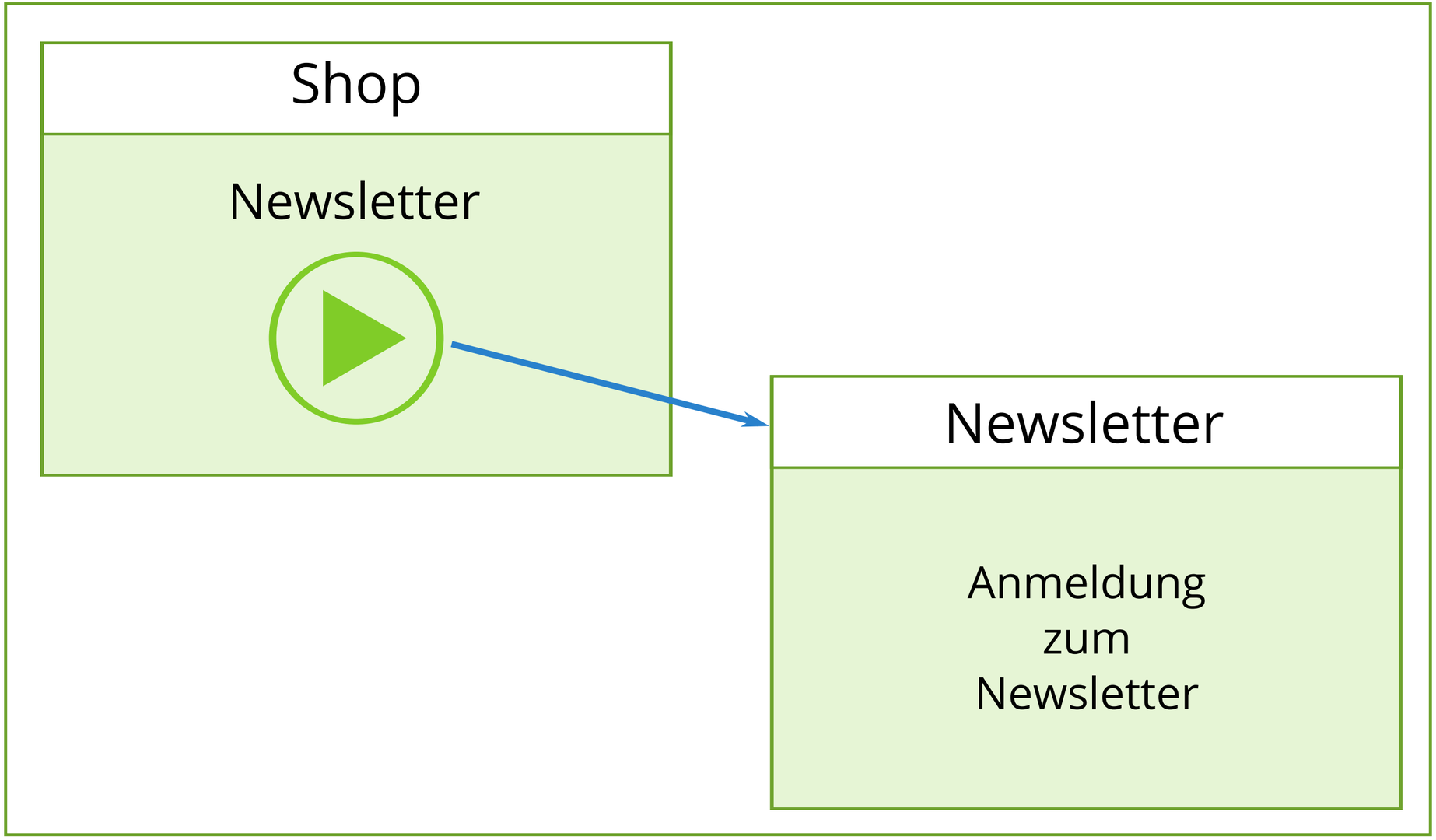
Tiefe Integration

- Aufruf von allgemeiner Funktionalität
  - Newsletter abbonieren
- Recherche / Information
  - Rechnungen eines Kunden
- Sprung zu spezifischem Usecase
  - Bestellung xy stornieren

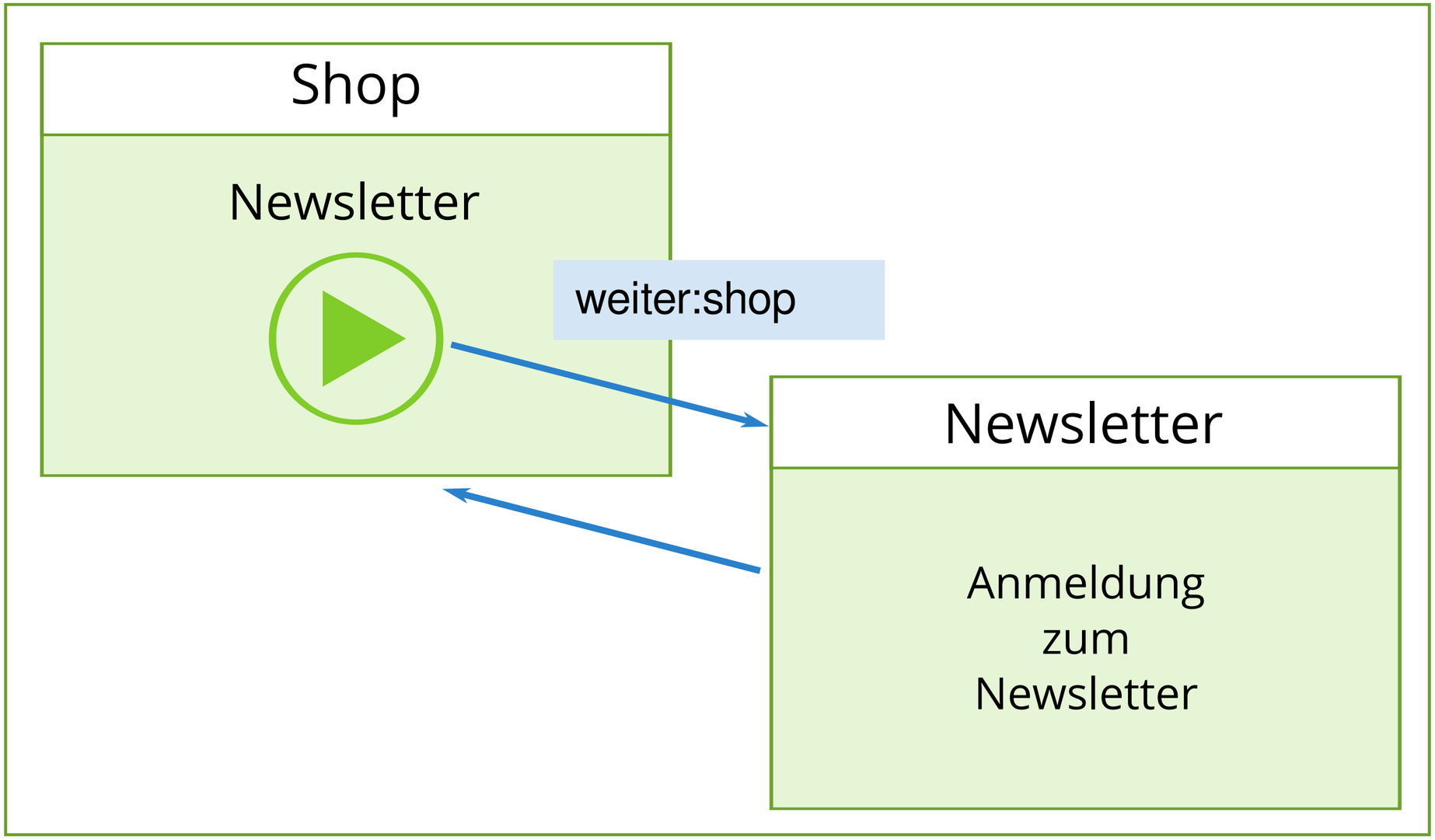
Optionen Tiefe Integration

- 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)

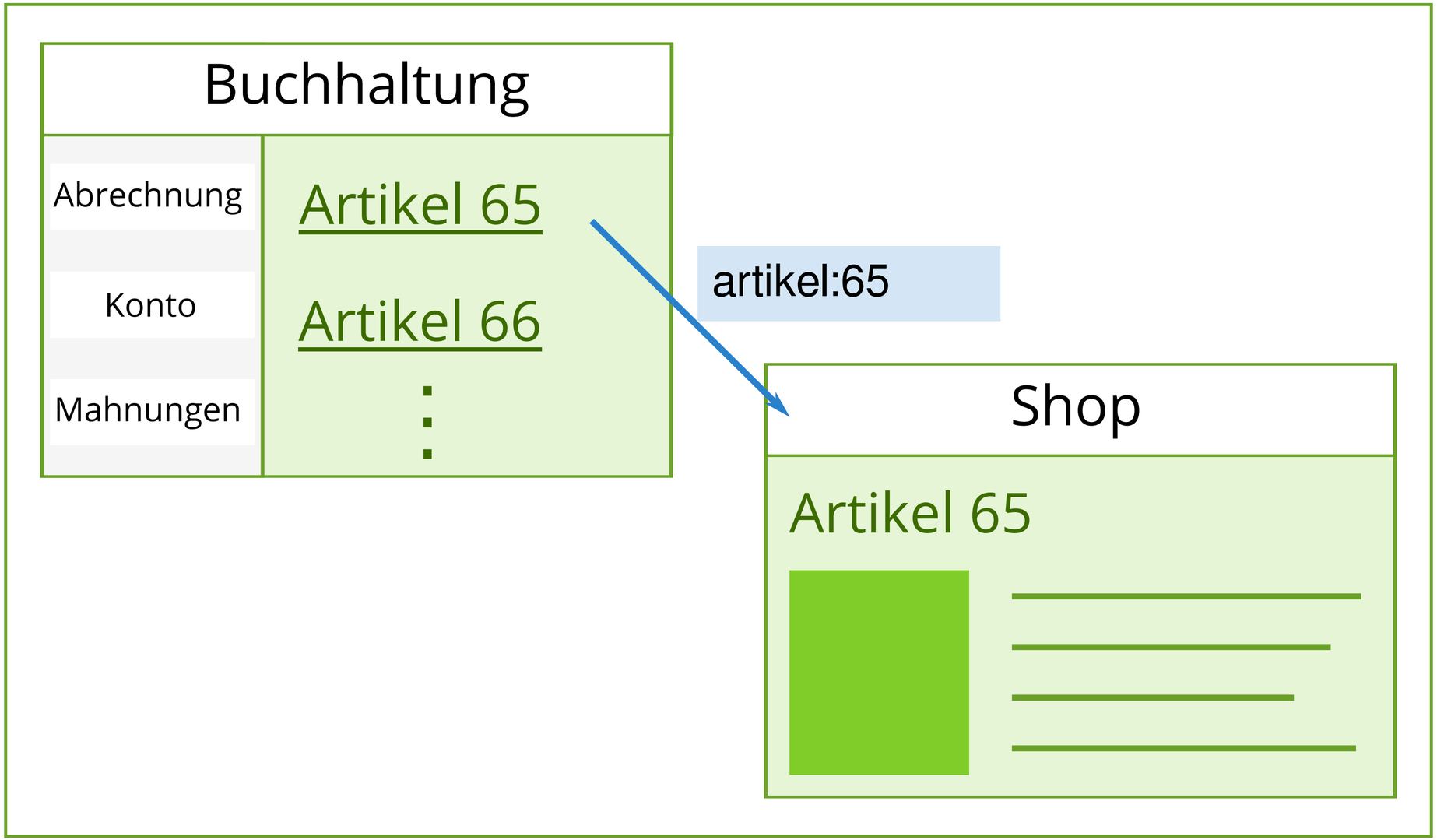
Link auf Unterbereich von SCS



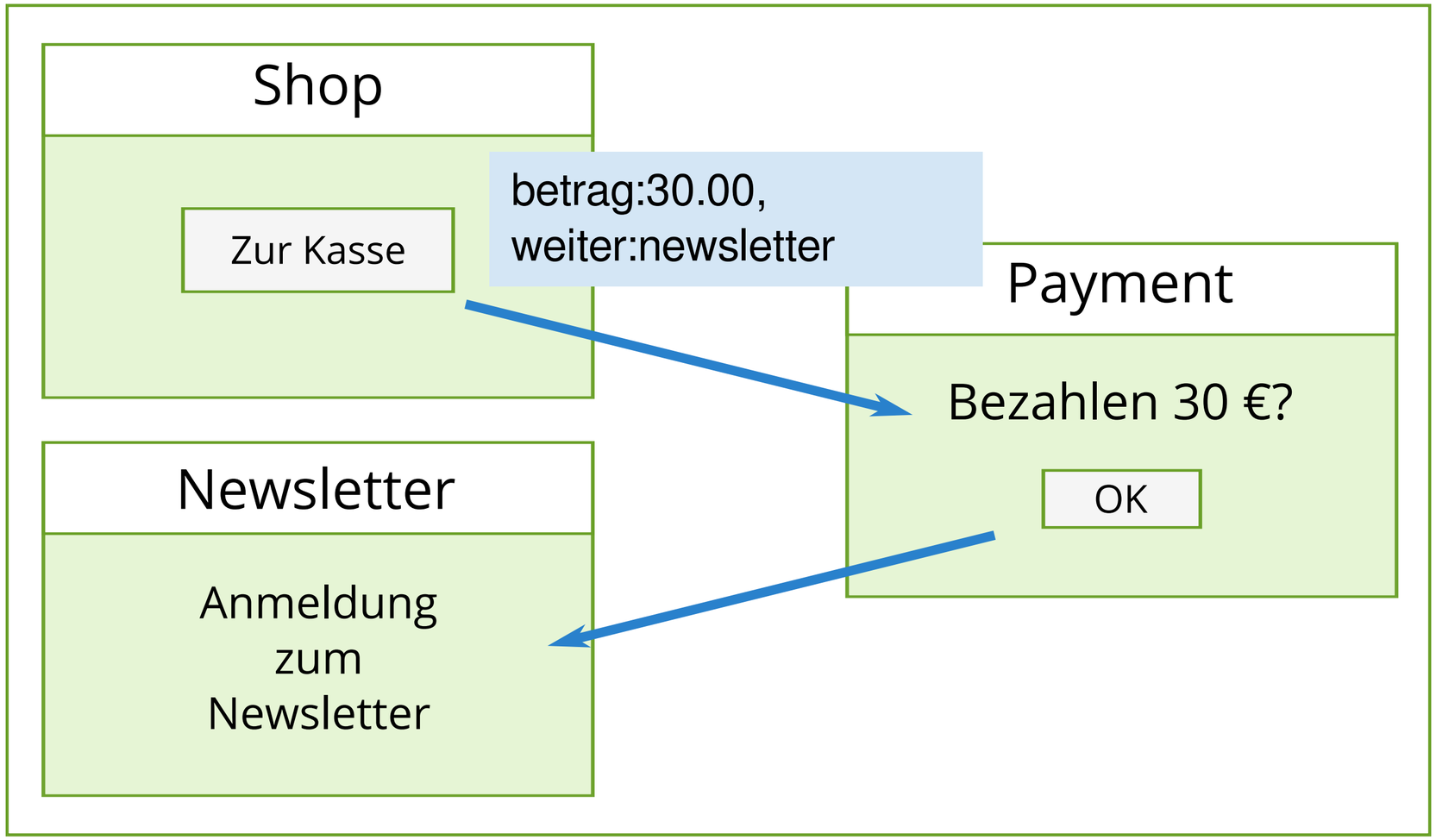
Link mit (Rück-)Sprungziel



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

Frontend-Technologie für SCS

- SCS sind Webanwendungen
- Webentwicklung im Enterprise-Umfeld verlangt:
  - Investitionssicherheit (Weiterentwicklung, Verbreitung, Community)
  - Hohe Entwicklerproduktivität
  - Gute Dokumentation
- Konsistenter Technologiestack nötig
  - Kann durch Framework bereitgestellt werden
  - Modern und kuratierter Stack

Server-Side

- JavaServer Faces
- Spring WebMVC
- Play
- Wicket

Client-Side (SPA)

- Angular
- React
- Polymer
- Vue
- (Vaadin / GWT)

Client-Side (SPA)

- SPA steht nicht im Widerspruch zu SCS
- SCS Architektur ist agnostisch bzgl. Umsetzung
- SCS führt zu regelmäßigem Wechsel des Systems durch Anwender
  - Schneller Anwendungsstart
  - Umgang mit lokalem Zustand
- Frühe Frameworks und Designs waren nicht optimal im SCS Kontext
  - Langsamer Anwendungsstart

Abwägung im Enterprise-Kontext

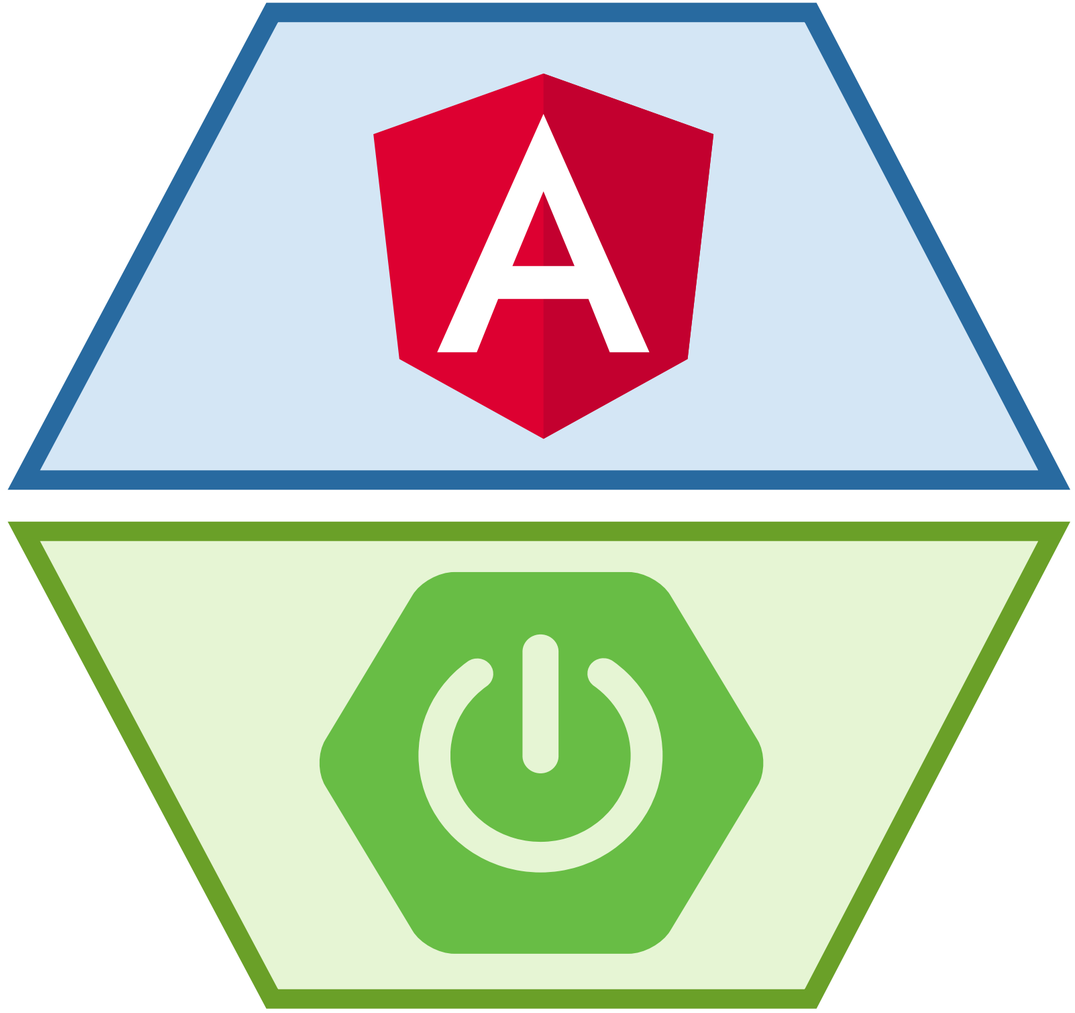
- Schwierig durch reine Server-Side-Stacks Desktop-Usability zu erzielen
- Angular + TypeScript
  - Modulsystem, DI
  - Typsicher (Refactoring/Wartung)
  - Für Java-Entwickler vertraute Muster
- Alternativen: React, Polymer, Vue
  - Liefern keinen vollständigen Stack, muss zusammengestellt werden

Demo

- Umsetzung mit Angular Frontend
  - Zwei SCS: Shop und Lager
  - Wechsel zwischen den Anwendungen über Link
  - Gemeinsame Navigation zur Build Time
  - Reload, Bookmarkfähigkeit
  - Umgang mit Zustand

SCS mit Angular

- Vorgehen repräsentativ für andere SPA Frameworks
- HTML5 Routing verwenden (Server-Side Rewrite Rules)
- Angular Modulsystem ermöglicht große, wartbare Anwendungen
  - Auch für komplexe SCS geeignet
- Angular Features: Crawlbar (SEO), I18N, Wiederverwendung, Wartbarkeit (TypeScript)
- Zustandsverwaltung mit @ngrx/store



Lager-Beispiel: Microservice mit Frontend-Anteil

- Technologien
  - Backend: Spring Boot
  - Frontend: Angular
- Build
  - Maven
  - NPM und Angular-CLI
- Deployment
  - Docker

Build Pipeline

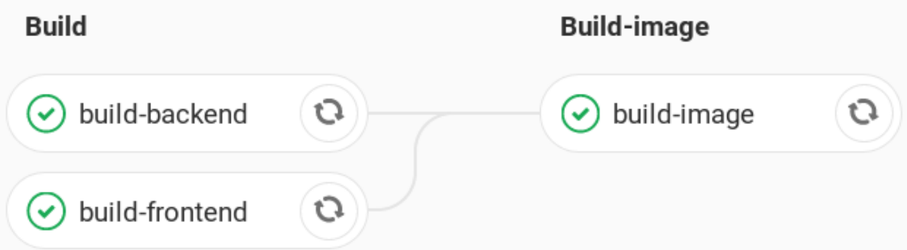
passed **Pipeline #62** triggered 50 minutes ago by **Stefan Reuter**

Build frontend and backend in parallel

3 jobs from `master` in 2 minutes 42 seconds (queued for 2 seconds)

`28c7fe0f` ...

Pipeline Jobs 3



Build mit CI Server

- Beispiel: GitLab CI
  - Stages: build und build-image (Docker)

```
stages:
  - build
  - build-image
variables:
  DIST_DIR: spring-boot-angular/frontend/dist/
  JAR_FILE: spring-boot-angular/backend/target/backend.jar
  DOCKERFILE: spring-boot-angular/Dockerfile.prod
  IMAGE: trion/scs-demo
```

CI Server: Backend

- Spring Boot Build mit Build-Container `maven`

```
build-backend:
  stage: build
  image: maven:3-jdk-8
  script:
    - cd spring-boot-angular/backend
    - mvn verify -B
  artifacts:
    expire_in: 1 day
    paths:
      - $JAR_FILE
  tags:
    - x86_64
```

CI Server: Frontend

- Frontend Build mit Build-Container `trion/ng-cli`

```
build-frontend:
  stage: build
  image: trion/ng-cli
  script:
    - cd spring-boot-angular/frontend
    - npm install
    - ng build --prod
  artifacts:
    expire_in: 1 day
    paths:
      - $DIST_DIR
  tags:
    - x86_64
```

CI Server: Docker Image

- Auslieferung des SCS Artefakts als Docker-Image

```
build-image:
  stage: build-image
  image: docker
  script:
    - docker build --build-arg JAR_FILE=$JAR_FILE --build-arg DIST_DIR=$DIST_DIR -t $IM
    - docker push $IMAGE
  tags:
    - x86_64
```

Deployment

- Microserver enthält das Frontend
  - Konsistenz durch gemeinsamen Build, Test und Deployment aller Bestandteile
- Optionen
  - Paketierung im JAR
  - In separatem Verzeichnis als statische Ressourcen
- Kombinierbar mit Docker in beiden Varianten

Reload und Bookmarks

- Anforderungen
  - Reload (F5)
  - Bookmarks (z.B. auf eine Artikeldetailseite)
  - Browser Navigation (Back/Forward)
- Umsetzung
  - Spring ResourceResolver liefert die Angular-Anwendung für alle Unterpfade aus
  - Angular übernimmt Route initial aus URL
  - Angular setzt URL auf Basis aktiver Route per HTML5 History API

Tipp: Namespace

- Zugriff auf Backend durch `/api`
  - Erleichtert HTTP Handling
  - Speziell bei Server-Side-Rendering (Crawling/ohne JS)

Tipp: Docker

- SCS Gedanke und Docker bringt Vorteile
- Für Build
  - Umgebung benötigt keine speziellen Werkzeuge
- Für Tests
  - Isolierter Test, kleinerer Test-Scope
  - Mit Docker eigener Datenbestand simpel
  - Reproduzierbare, stabile Tests

Häufige Fragen zur Umsetzung

Was ist mit …

- Globalem Look-and-Feel
  - UI: CSS Asset sharing, Redundante Implementierung von CI Guidelines
  - UX: Abstimmung
  - Gefahr, dass Autonomie verloren geht

Was ist mit ...

- Microfrontends
  - Custom Framework
  - Technologische Kopplung
  - Vergleichbar mit Portal-Server
  - Deutet auf nicht gelungenen Domänen-Schnitt
  - Evtl. passt hier SCS Architektur nicht

Was ist mit …

- Transklusion (SSI, ESI, Ajax-Fragments)
  - Nur bei autonomen Anteilen
  - Sekundäre Inhalte, Read-Only
  - Gefahr von Laufzeit-Monolith
  - Gefahr von hoher Komplexität
  - Umfangreicher Einsatz deutet auf nicht gelungenen Domänen-Schnitt
  - Security Anforderungen schwierig umzusetzen

Was ist mit …

- Nativen Anwendungen (Desktop, iOS, Android)
  - HTML-Hybrid kann mit SCS harmonisieren
  - SCS nicht dogmatisch sehen

Fazit

- SCS funktioniert mit modernen Frameworks und Konzepten
  - Gute Erfahrungen mit Angular, Spring Boot und Kafka
- Herausforderungen
  - Guter Schnitt essentiell
  - (Re-)Integration für Anwender
  - Organisatorische Autonomie
- Bei hochintegrierten Frontendanwendungen passt SCS-Ansatz evtl. nicht optimal
  - Monolithisches Frontend stellt höhere Anforderungen an Entwickler und Framework, mit Angular handhabbar

danke.

- Stefan Reuter (@stefanreuter)
- Thomas Kruse (@everflux)