Feature Toggles vs. Clean Code

Featuremanagement an HATEOAS konformen REST-Schnittstellen

Jörg Adler, 30.9.2016

Mercateo ist Europas führende Beschaffungsplattform.

- $>$ 1 Mio. Geschäftskunden
- 21 Mio. Artikel
- mehr als 10.000 Hersteller

Mercateo ist Europas führende Beschaffungsplattform.

- $>$ 1 Mio. Geschäftskunden
- 21 Mio. Artikel
- mehr als 10.000 Hersteller
- Entwicklung von der Beschaffungsplattform zur Transaktionsplattform
- Massive Veränderungen in der IT-Landschaft
- hin zu REST und asynchronen Schnittstellen

Jörg Adler
joerg.adler@mercateo.com

Inhaltsverzeichnis

Definition

Feature Toggle (von engl. feature = Eigenschaft, Fähigkeit, Funktion; Toggle = Schalter) ist eine Programmiertechnik in der modernen Softwareentwicklung, bei der ein in der Entwicklung befindliches Feature oder eine Funktionalität zur Laufzeit der Software an- oder ausgeschaltet werden kann.[1]

[1] https://de.wikipedia.org/wiki/Feature_Toggle

Anwendungsbeispiele

- zeitliches Entkoppeln Deployment und Release

[1] http://martinfowler.com/articles/feature-toggles.html

Anwendungsbeispiele

- zeitliches Entkoppeln Deployment und Release
- A/B Testing, Canary Releases.
- Champagne Brunch [1]
- Alpha/Beta-Test für bestimmte Nutzer

[1] http://martinfowler.com/articles/feature-toggles.html

Anwendungsbeispiele

- zeitliches Entkoppeln Deployment und Release
- A/B Testing, Canary Releases.
- Champagne Brunch [1]
- Alpha/Beta-Test für bestimmte Nutzer
- Ops Toggles ("Notfall-Aus")
- unfertigen Code verstecken

[1] http://martinfowler.com/articles/feature-toggles.html

Kosten

- Komplexität im Code

Kosten

- Komplexität im Code
- Komplexität in Tests

Kosten

- Komplexität im Code
- Komplexität in Tests
- Entwicklungszeit für Ein- und **Ausbau**

Kosten

- Komplexität im Code
- Komplexität in Tests
- Entwicklungszeit für Ein- und **Ausbau**
- zusätzliche Infrastruktur

Setting the Stage

- System für Orders ("RESTful")

Setting the Stage

- System für Orders ("RESTful")
- neue Anforderung (TICKET-5):
  - verschickte Orders können wieder zurück gesandt werden
  - Nachricht soll mit verschickt werden
  - neues Feature soll erst an einigen Nutzern ausgetestet werden

Setting the Stage

- System für Orders ("RESTful")
- neue Anforderung (TICKET-5):
  - verschickte Orders können wieder zurück gesandt werden
  - Nachricht soll mit verschickt werden
  - neues Feature soll erst an einigen Nutzern ausgetestet werden
- neue Anforderung (TICKET-6):
  - Kunden sollen ihren bevorzugten Transportdienstleister angeben können
  - neues Feature soll erst an einigen Nutzern ausgetestet werden

Straight-Forward-Ansatz Server

- neue Resource um Feature-Daten abzurufen

Straight-Forward-Ansatz Server

- neue Resource um Feature-Daten abzurufen
- die neue Schnittstelle:

```
1   @Path("orders")
2   public class OrdersResource {
3     @Inject
4     private EnabledFeatures features;
5     ...
6
7     @Path("{orderId}/send-back")
8     @POST
9     @Consumes(MediaType.APPLICATION_JSON)
10    public void sendBack(
11          @PathParam("orderId") String orderId, SendBackJson sendBackJson) {
12      if(features.isTicket5Enabled()){
13          ...
14          CARRIER carrier = CARRIER.C1;
15          if(features.isTicket6Enabled()){
16               carrier = sendBackJson.getCarrier();
17          }...
18          orderService.sendBack(sendBackDto, legalEntityId);
19      }
20      else{ /*throw some exception*/ }
21  }
```

Straight-Forward-Ansatz Server

- neue Resource um Feature-Daten abzurufen
- die neue Schnittstelle:

```
1   @Path("orders")
2   public class OrdersResource {
3     @Inject
4     private EnabledFeatures features;
5     ...
6
7     @Path("{orderId}/send-back")
8     @POST
9     @Consumes(MediaType.APPLICATION_JSON)
10    public void sendBack(
11          @PathParam("orderId") String orderId, SendBackJson sendBackJson) {
12      if(features.isTicket5Enabled()){
13          ...
14          CARRIER carrier = CARRIER.C1;
15          if(features.isTicket6Enabled()){
16               carrier = sendBackJson.getCarrier();
17          }...
18          orderService.sendBack(sendBackDto, legalEntityId);
19      }
20      else{ /*throw some exception*/ }
21  }
```

- natürlich noch verbesserbar (Strategy-Pattern...)

Straight-Forward-Ansatz Client

- zusätzlicher Call, um Features abzurufen (blockierend!)

Straight-Forward-Ansatz Client

- zusätzlicher Call, um Features abzurufen (blockierend!)
- Formular

```
<form ng-show="(feature.isTicket5Enabled()) && (order.state=='SHIPPED')">
    ...
    <input ng-show="feature.isTicket6Enabled" />
</form>
```

Straight-Forward-Ansatz Client

- zusätzlicher Call, um Features abzurufen (blockierend!)
- Formular

```
<form ng-show="(feature.isTicket5Enabled()) && (order.state=='SHIPPED')">
    ...
    <input ng-show="feature.isTicket6Enabled" />
</form>
```

- auch hier Verbesserung durch verschiedene Pattern

Straight-Forward-Ansatz Client

- zusätzlicher Call, um Features abzurufen (blockierend!)
- Formular

```
<form ng-show="(feature.isTicket5Enabled()) && (order.state=='SHIPPED')">
    ...
    <input ng-show="feature.isTicket6Enabled" />
</form>
```

- auch hier Verbesserung durch verschiedene Pattern
- Verdopplung von Code, verlangsamen der Laufzeit

Hypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate

Hypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate

- integraler Bestandteil von REST

Hypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate

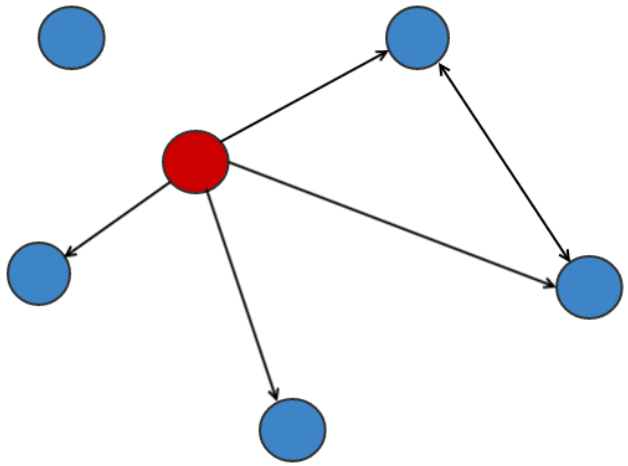
- integraler Bestandteil von REST
- Zustandsübergänge am Client ausschließlich über Links vom Server

Hypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate

- integraler Bestandteil von REST
- Zustandsübergänge am Client ausschließlich über Links vom Server
- Links kommen nur **dynamisch** vom Server

Hypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate

- integraler Bestandteil von REST
- Zustandsübergänge am Client ausschließlich über Links vom Server
- Links kommen nur **dynamisch** vom Server
- im Client nur eine Einsprungs-URL und das Wissen über die Linknamen



„Software design on the scale of decades: every detail is intended to promote software longevity and independent evolution. Many of the constraints are directly opposed to short-term efficiency. Unfortunately, people are fairly good at short-term design, and usually awful at long-term design.“

Fielding, Roy T. “REST APIs must be hypertext-driven“

HATEOAS bei Mercateo

LDO - Link Description Object

```
1  {
2    "rel": "send-back",
3    "method": "POST",
4    "href": "http://example.org/api/orders/1/send-back",
5    "targetSchema": {},
6    "schema": {} // auf der nächsten Seite
7  }
```

send-back Schema

```
1  {
2    "type": "object",
3    "properties": {
4      "message": {
5        "type": "string"
6      }
7    },
8    "required": [ "message" ]
9  }
```

Wie wird daraus eine link-basierte REST API ?

Einsprungspunkt

```
{
  "_schema": {
    "links": [
      // ...
      {
        "rel": "orders",
        "method": "GET",
        "href": "http://example.org/api/orders"
      }
      // ...
    ]
  },
  "firstName": "Max",
  "lastName": "Mustermann"
}
```

http://example.org/api/orders/

```
{
  "_schema": {
    "links": [
      { /* create-LDO */ }
    ]
  },
  "members": [
    {
      "_schema": {
        "links":[
          { /* cancel-LDO */ }
        ]
      },
      "id": 1,
      "total": 334.4,
      "state": "OPEN" // || PROCESSING || SHIPPED || CANCELED || RETURNED
    },
    {
      "_schema": {
        "links":[
          { /* send-back-LDO */ }
        ]
      },
      "id": 2,
      "total": 200.0,
      "state": "SHIPPED" // || PROCESSING || OPEN || CANCELED || RETURNED
    }
  ]
}
```

Client-Anforderungen

- prüfen ob die Links vorhanden sind, denen er folgen will

Client-Anforderungen

- prüfen ob die Links vorhanden sind, denen er folgen will
- prüfen ob Schema nach seinen Vorstellungen konform ist

Client-Anforderungen

- prüfen ob die Links vorhanden sind, denen er folgen will
- prüfen ob Schema nach seinen Vorstellungen konform ist
- Beispiel Formular (Links sind in Fields umgewandelt!)

```
<form ng-show="response.$sendBack">
    ...
    <input ng-show="resonse.$sendBack._schema.preferredCarrier"/>
</form>
```

Was hat dies mit FeaturenToggles zu tun?

Links I

- Hinzufügen oder Entfernen von Links ist unkritisch

Links I

- Hinzufügen oder Entfernen von Links ist unkritisch
- Feature Toggle schaltet somit nur den Link an und aus

Hyper-Schema

```
1  {
2     ...
3     "members": [
4       {
5         "_schema": {
6           "links":[
7           ]
8         },
9         "id": 1,
10        "total": 334.4,
11        "state": "OPEN"
12      },
13      {
14        "_schema": {
15          "links":[
16
17          ]
18        },
19        "id": 2,
20        "total": 200.0,
21        "state": "SHIPPED"
22      }
23    ]
24 }
```

Hyper-Schema

```
{
  ...
  "members": [
    {
      "_schema": {
        "links":[
        ]
      },
      "id": 1,
      "total": 334.4,
      "state": "OPEN"
    },
    {
      "_schema": {
        "links":[

        ]
      },
      "id": 2,
      "total": 200.0,
      "state": "SHIPPED"
    }
  ]
}
```

mit TICKET-5

```
{
  ...
  "members": [
    {
      "_schema": {
        "links":[
        ]
      },
      "id": 1,
      "total": 334.4,
      "state": "OPEN"
    },
    {
      "_schema": {
        "links":[
          { /* send-back-LDO */ }
        ]
      },
      "id": 2,
      "total": 200.0,
      "state": "SHIPPED"
    }
  ]
}
```

JSON Schema I

send-back Schema

```
1  {
2    "type": "object",
3    "properties": {
4      "message": {
5        "type": "string"
6      }
7    },
8    "required": [ "message" ]
9  }
```

JSON Schema I

send-back Schema

```
1  {
2    "type": "object",
3    "properties": {
4      "message": {
5        "type": "string"
6      }
7    },
8    "required": [ "message" ]
9  }
```

mit TICKET-6

```
1  {
2    "type": "object",
3    "properties": {
4      "message": {
5        "type": "string"
6      },
7      "preferredCarrier": {
8        "type": "string",
9        "enum": [ "C1" , "C2" ]
10     }
11   },
12   "required": [ "message" ]
13 }
```

JSON Schema II

- Komplizierter, da inkompatible Änderungen möglich!

JSON Schema II

- Komplizierter, da inkompatible Änderungen möglich!
- Client kann das Schema mitsenden
- Server entscheidet dann ob er den Request verarbeitet

rest-schemagen

- OSS Aufsatz auf Jersey zur dynamischen Link-Erzeugung

rest-schemagen

- OSS Aufsatz auf Jersey zur dynamischen Link-Erzeugung
- Mitarbeit ist sehr erwünscht :-)

rest-schemagen

- OSS Aufsatz auf Jersey zur dynamischen Link-Erzeugung
- Mitarbeit ist sehr erwünscht :-)
- Github: `https://github.com/Mercateo/rest-schemagen`
- Maven: `http://search.maven.org/#search%7Cga%7C1%7Ca%3A%22common.rest.schemagen%22`
- Beispiel-Projekt: `https://github.com/Mercateo/rest-demo-feature`

Aufgabe

- Überblick über bestehende Softwaremodule für die Integration von REST-Hyperlinks.

Jersey Declarative Hyperlinking – Beispiel

```
1   @Data
2   @Builder
3   public class OrderWrapper {
4
5     @InjectLinks({...
6       @InjectLink(resource = OrdersLinkingResource.class, rel = "send-back",
7         method = "sendBack",
8         condition = "${resource.featureChecker.ticket_5}",
9         bindings = {@Binding(name = "orderId", value = "${instance.order.id}")})
10      ...})
11      @XmlJavaTypeAdapter(Link.JaxbAdapter.class)
12      List<Link> links;
13
14      @JsonUnwrapped
15      private OrderJson order;
16  }
```

- Annotationen an den Response-Klassen
- Kontrolle des Verhaltens über Expression-Language

Jersey Declarative Hyperlinking – Fazit

- Methoden-Referenz über String verlässt Code-Ebene

Jersey Declarative Hyperlinking – Fazit

- Methoden-Referenz über String verlässt Code-Ebene
- Expressions können nur auf Inhalte des Response-Objekts und der Ressource zugreifen
- Bindings können nur auf Inhalte des Response-Objekts und der Ressource zugreifen

Jersey Declarative Hyperlinking – Fazit

- Methoden-Referenz über String verlässt Code-Ebene
- Expressions können nur auf Inhalte des Response-Objekts und der Ressource zugreifen
- Bindings können nur auf Inhalte des Response-Objekts und der Ressource zugreifen
- nur mit Jersey-Testframework testbar.

Link auf Methoden

```
1     Link link = linkTo(methodOn(OrdersResource.class).sendBack("1")).
2                   withRel("send-back");
```

Link auf Entities

```
1     EntityLinks links = ...
2     LinkBuilder builder = links.linkFor(OrdersResource.class);
3     Link link = links.linkToSingleResource(OrdersResource.class, 1L);
```

Spring Hateoas – Fazit

- Parameter der Methoden-Referenz durch Proxy-Aufruf bestimmt.
- Arbeitet nur mit Typen, die nicht `final` sind.
- Links sind nicht dynamisch erzeugbar.
- HAL-Format ohne HTTP-Verben

Weiter Anforderungen

- Qualität/Existenz des integrierten Schemagenerators

Weiter Anforderungen

- Qualität/Existenz des integrierten Schemagenerators
- Link-Erzeugung dynamisch

Weiter Anforderungen

- Qualität/Existenz des integrierten Schemagenerators
- Link-Erzeugung dynamisch
- Unterstützung für Pfade und Subressourcen

Beispielimplementierung Jersey

Interfaces

- beliebiges Feature-Toggle Framework einbinden

Interfaces

- beliebiges Feature-Toggle Framework einbinden
- Feature - Annotation schreiben/einbinden
- Für Links:

```
1  public interface MethodCheckerForLink extends Predicate<Scope> {
2      static MethodCheckerForLink fromPredicate(Predicate<Scope> predicate) {
3          return s -> predicate.test(s);
4      }
5  }
6  public abstract class Scope {
7      private final Class<?> invokedClass;
8      private final Method invokedMethod;
9      private final Object[] params;...
10 }
```

Interfaces

- beliebiges Feature-Toggle Framework einbinden
- Feature - Annotation schreiben/einbinden
- Für Links:

```
1  public interface MethodCheckerForLink extends Predicate<Scope> {
2      static MethodCheckerForLink fromPredicate(Predicate<Scope> predicate) {
3          return s -> predicate.test(s);
4      }
5  }
6  public abstract class Scope {
7      private final Class<?> invokedClass;
8      private final Method invokedMethod;
9      private final Object[] params;...
10 }
```

- für Schemata:

```
1  public interface FieldCheckerForSchema extends BiPredicate<Field, CallContext> {
2  }
```

Interfaces

- beliebiges Feature-Toggle Framework einbinden
- Feature - Annotation schreiben/einbinden
- Für Links:

```
1   public interface MethodCheckerForLink extends Predicate<Scope> {
2       static MethodCheckerForLink fromPredicate(Predicate<Scope> predicate) {
3           return s -> predicate.test(s);
4       }
5   }
6   public abstract class Scope {
7       private final Class<?> invokedClass;
8       private final Method invokedMethod;
9       private final Object[] params;...
10  }
```

- für Schemata:

```
1   public interface FieldCheckerForSchema extends BiPredicate<Field, CallContext> {
2   }
```

- Filter/Interceptor für die Feature Annotation

Ressource

```
1   @Path("orders")
2   public class OrdersResource {
3     ...
4
5
6
7
8     @Path("{orderId}/send-back")
9     @POST
10    @Consumes(MediaType.APPLICATION_JSON)
11
12    public void sendBack(
13          @PathParam("orderId") String orderId, SendBackJson sendBackJson) {
14
15        ...
16
17
18  }
```

Ressource

```
1   @Path("orders")
2   public class OrdersResource {
3     ...
4
5
6
7
8     @Path("{orderId}/send-back")
9     @POST
10    @Consumes(MediaType.APPLICATION_JSON)
11    @Feature(TICKET_5)
12    public void sendBack(
13          @PathParam("orderId") String orderId, SendBackJson sendBackJson) {
14
15        ...
16
17
18  }
```

Ressource

```
1   @Path("orders")
2   public class OrdersResource {
3     @Inject
4     private OrderService orderService;
5     ...
6
7
8     @Path("{orderId}/send-back")
9     @POST
10    @Consumes(MediaType.APPLICATION_JSON)
11    @Feature(TICKET_5)
12    public void sendBack(
13          @PathParam("orderId") String orderId, SendBackJson sendBackJson) {
14
15      LegalEntityId legalEntityId = ...
16      orderService.sendBack(sendBackJson, legalEntityId);
17      ...
18  }
```

Kein Conditional

```
1   @JsonInclude(Include.NON_NULL)
2   @JsonIgnoreProperties(ignoreUnknown = true)
3   @Data
4   public class SendBackJson {
5
6       @NotNull
7       private String message;
8
9       @Feature(TICKET_6)
10      @DefaultValue("C1")
11      private CARRIER preferredCarrier;
12  }
```

Linkerzeugung I

```
1   ...
2   @Inject
3   private LinkMetaFactory linkMetaFactory;
4   ...
5   private Optional<Link> createSendBackLink(String id) {
6
7     LinkFactory<OrdersRessource> linkFactory =
8           linkMetaFactory.createFor(OrdersRessource.class);
9
10    ...
11
12  }
```

Linkerzeugung II

```
1   ...
2   @Inject
3   private LinkMetaFactory linkMetaFactory;
4   ...
5   private Optional<Link> createSendBackLink(String id) {
6
7     LinkFactory<OrdersRessource> linkFactory =
8           linkMetaFactory.createFor(OrdersRessource.class);
9
10     return linkFactory.forCall(Relation.of("send-back", RelType.INHERITED),
11          r -> r.sendBack(id, null));
12  }
```

Linkerzeugung

- IDE-Mittel (Call-Hierarchy,...) wieder nutzbar

Linkerzeugung

- IDE-Mittel (Call-Hierarchy,...) wieder nutzbar
- Features nur an einer Stelle

Linkerzeugung

- IDE-Mittel (Call-Hierarchy,...) wieder nutzbar
- Features nur an einer Stelle
- Schemaerzeugung nach einheitlichem Vorgehen

Tests

- Tests sehr einfach und performant
- Unit-Tests ohne Feature Toggles.

Tests

- Tests sehr einfach und performant
- Unit-Tests ohne Feature Toggles.
- vollständiger Verzicht auf Jersey-Tests als Unit-Test

Tests

- Tests sehr einfach und performant
- Unit-Tests ohne Feature Toggles.
- vollständiger Verzicht auf Jersey-Tests als Unit-Test
- Integrationstests auch weitestgehend ohne Mehraufwand möglich

Take-Aways

- HATEOAS nicht nur für die Entkopplung der URLs

Take-Aways

- HATEOAS nicht nur für die Entkopplung der URLs
- Am Server wird die Übersichtlichkeit massiv erhöht

Take-Aways

- HATEOAS nicht nur für die Entkopplung der URLs
- Am Server wird die Übersichtlichkeit massiv erhöht
- Am Client nur noch Code für die Darstellung

Take-Aways

- HATEOAS nicht nur für die Entkopplung der URLs
- Am Server wird die Übersichtlichkeit massiv erhöht
- Am Client nur noch Code für die Darstellung
- Somit sind die Vorteile nicht erst nach Jahrzehnten sichtbar.

Take-Aways

- HATEOAS nicht nur für die Entkopplung der URLs
- Am Server wird die Übersichtlichkeit massiv erhöht
- Am Client nur noch Code für die Darstellung
- Somit sind die Vorteile nicht erst nach Jahrzehnten sichtbar.
- Github: `https://github.com/Mercateo/rest-schemagen`

Vielen Dank für Ihre Aufmerksamkeit.

Fragen? :)