ELO Flows Development

Flow Komponente

Inhaltsverzeichnis

Flow Komponente

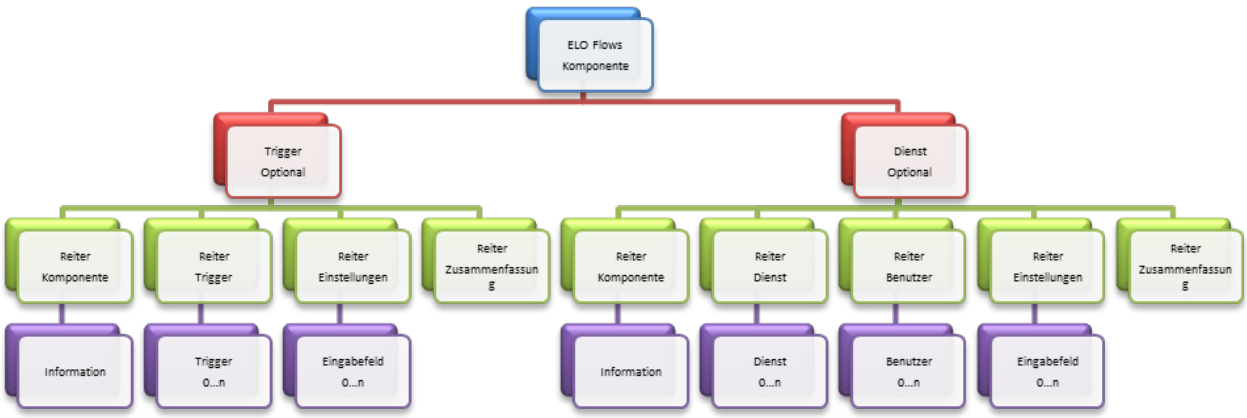
Aufbau der Komponente

Eine ELO Flows-Komponente kann zunächst einen oder mehrere Trigger, einen oder mehrere Dienste oder beides zur Verfügung stellen. Alle Trigger und Dienste können konfiguriert werden. Über Schlüssel können Daten zwischen den Komponenten ausgetauscht werden. Jede Komponente stellt für ihre Trigger und Dienste umfangreiche Informationen für das Handling bereit.

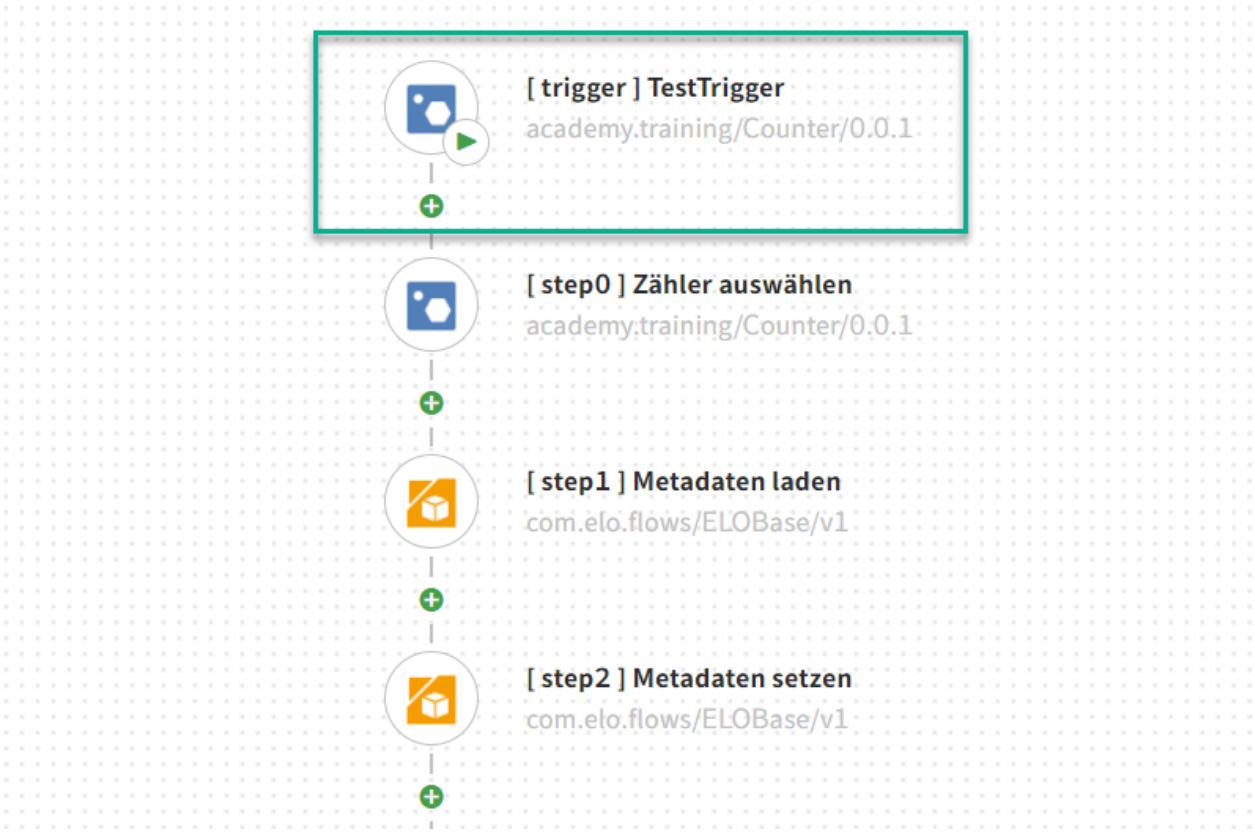
Die Konfiguration der Komponente erfolgt über eine grafische Benutzerschnittstelle. Hier können individuelle Einstellungen vorgenommen werden. Die programmtechnische Umsetzung wird in Java und hier insbesondere über die Annotationen realisiert.

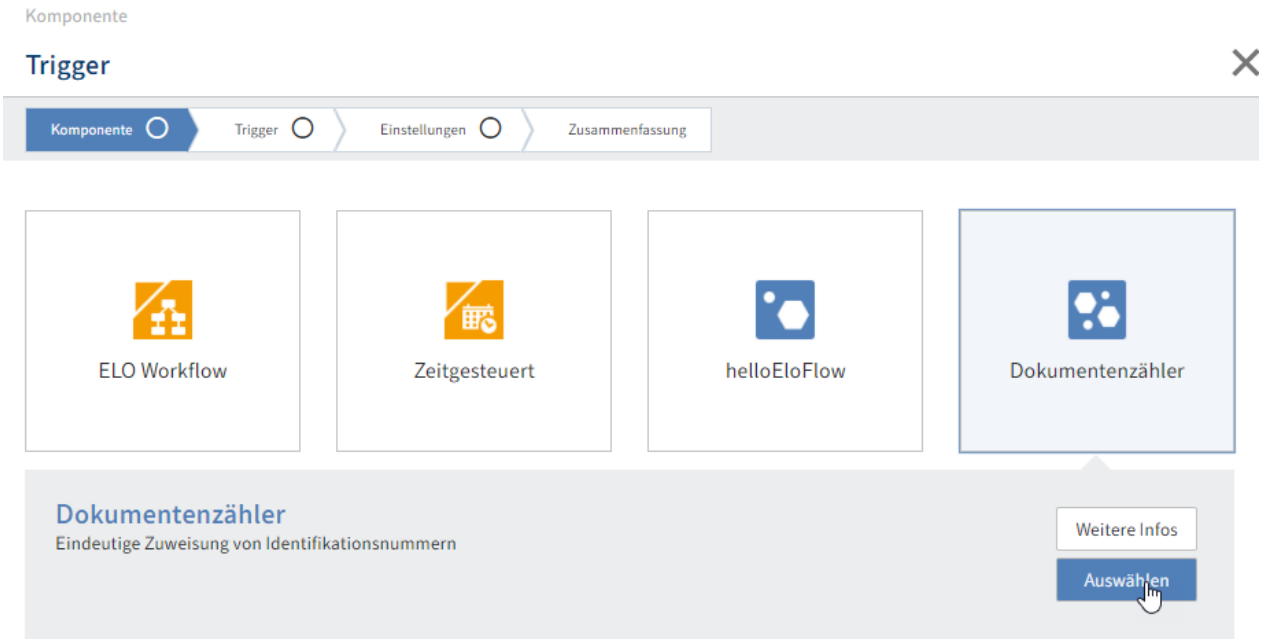
Der Austausch der Daten findet über ein internes JSON-Konfigurationsobjekt statt. Die Daten werden über In- und Output-Klassen implementiert.

Die Funktionsbeschreibungen der Komponenten werden durch die Anbindung von *Markdown-Dateien (.md)* hinterlegt.

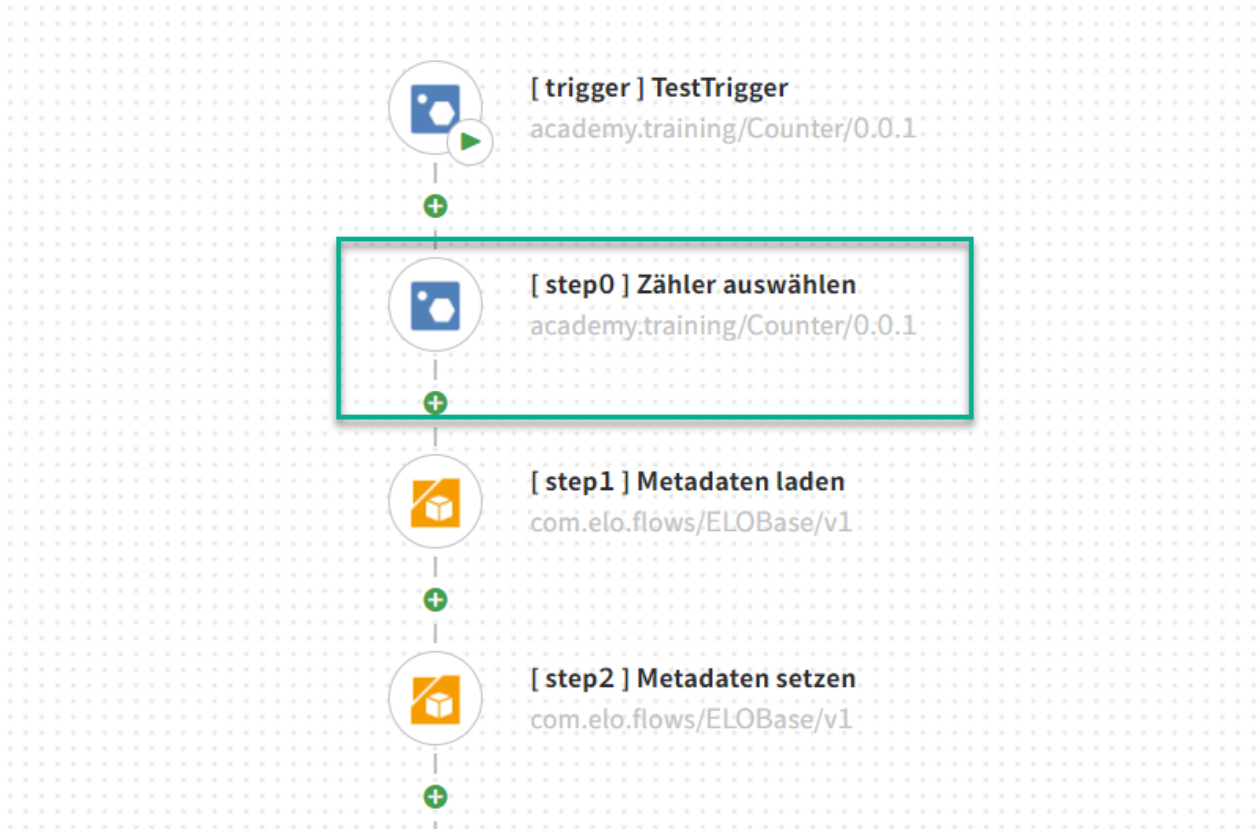


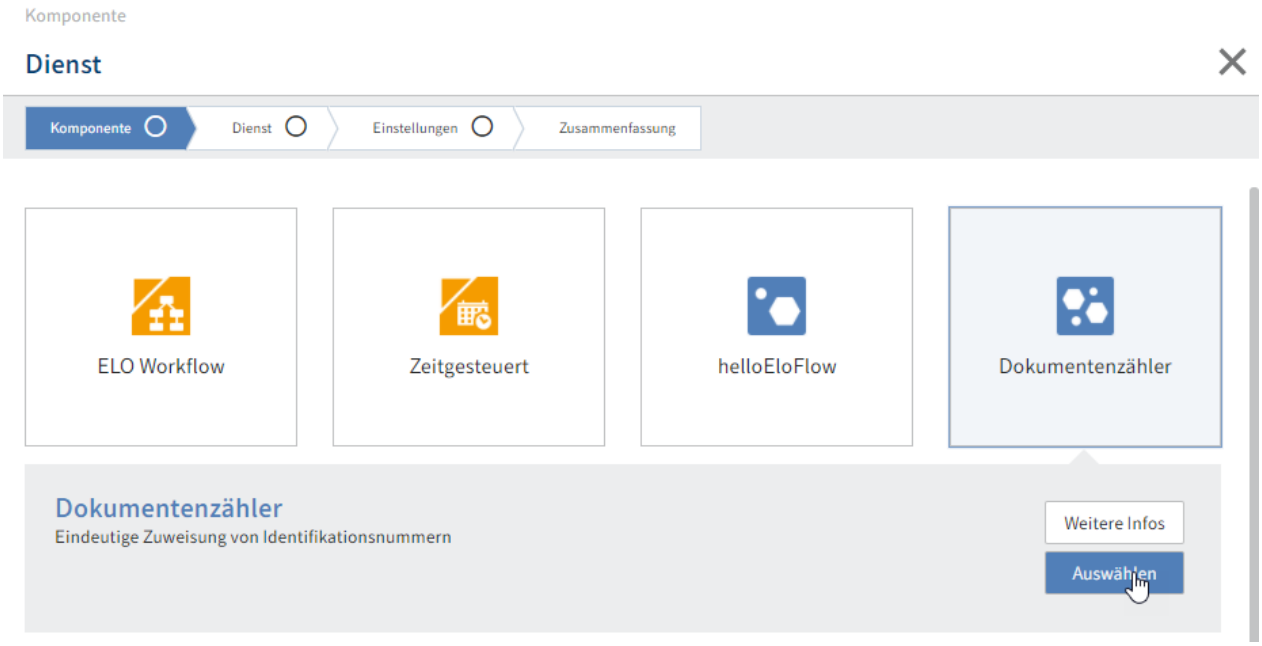
Implementierung eines Triggers.



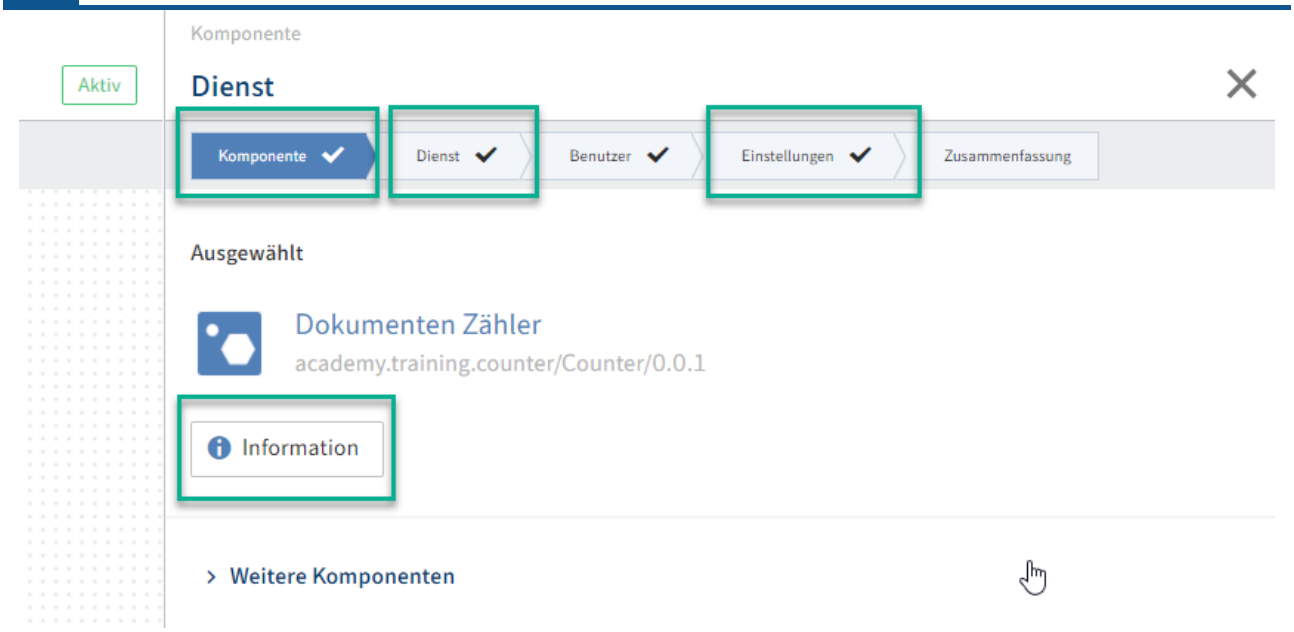


Implementierung eines Dienstes.

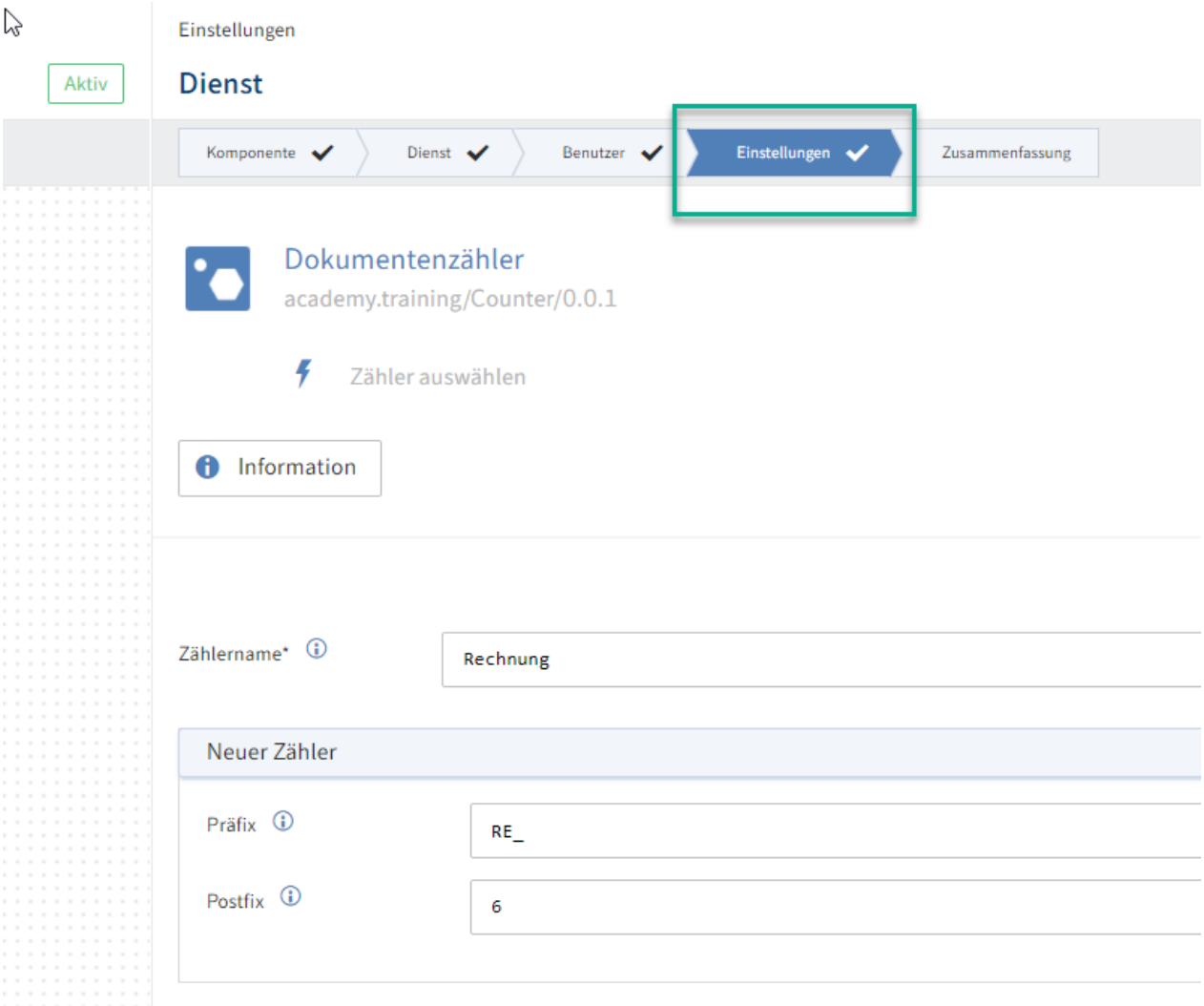




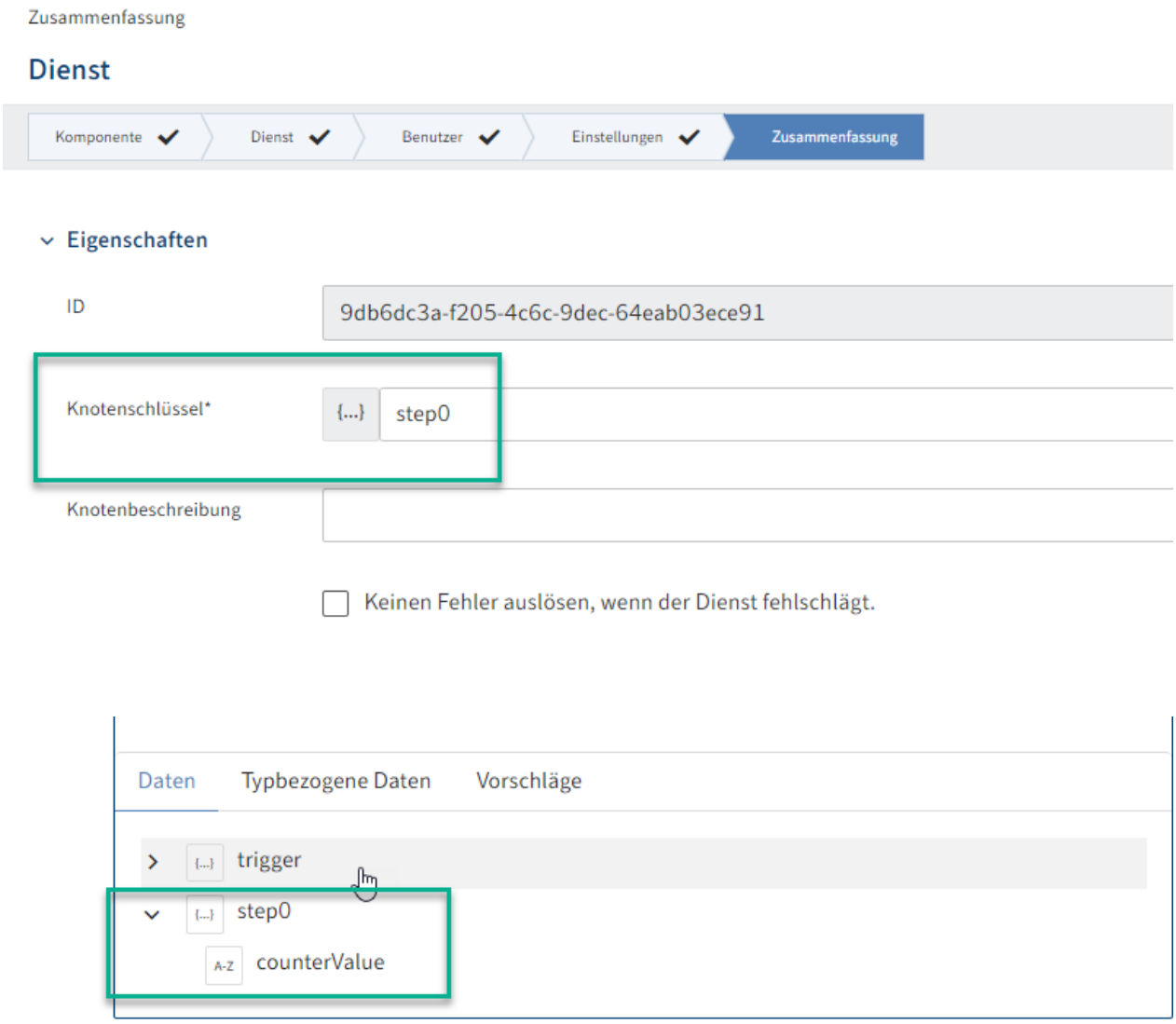
Implementierungsmöglichkeiten innerhalb einer Komponente.



Implementierungsmöglichkeiten innerhalb des Tabs Einstellungen.

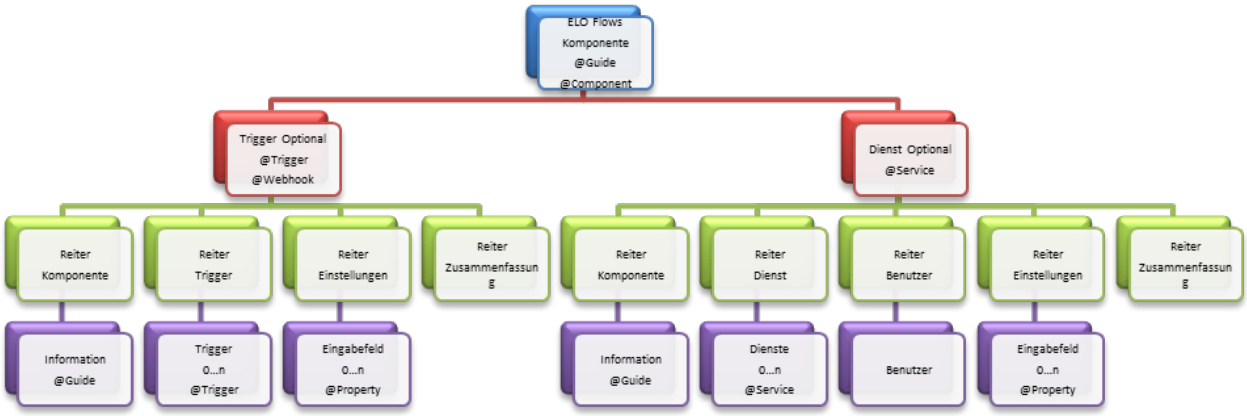


In einem JSON-Konfigurationsobjekt können über Schlüssel, Daten an nachfolgende Komponenten weitergereicht werden.


Zusammenfassung
Dienst
Komponente
Dienst
Benutzer
Einstellungen
Zusammenfassung
Eigenschaften
ID
9db6dc3a-f205-4c6c-9dec-64eab03ece91
Knotenschlüssel*
{...}
step0
Knotenbeschreibung
Keinen Fehler auslösen, wenn der Dienst fehlschlägt.
Daten
Typbezogene Daten
Vorschläge
trigger
step0
counterValue

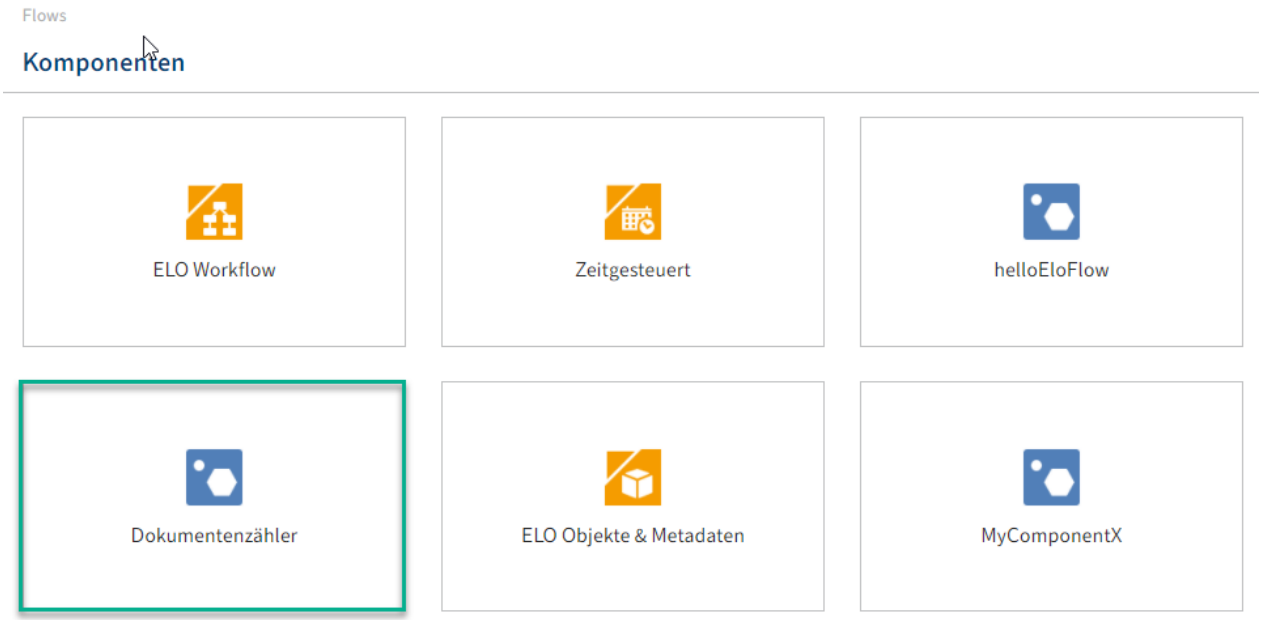
Übersicht ELO Flows-Komponenten

An dieser Stelle sind alle grafischen Komponenten mit den dazugehörigen Annotationen in jeweils einem Beispiel aufgeführt.



ELO Flows-Komponente

Jeder Flow basiert auf Komponenten. Komponenten können mehrere Elemente enthalten. Beispielsweise stellt eine Komponente einen oder mehrere Trigger zur Verfügung. Diese Trigger lösen dann weitere Aktionen im Flow aus. Diese Aktionen werden über Dienste realisiert. Die Dienste werden ebenfalls von Komponenten bereitgestellt.



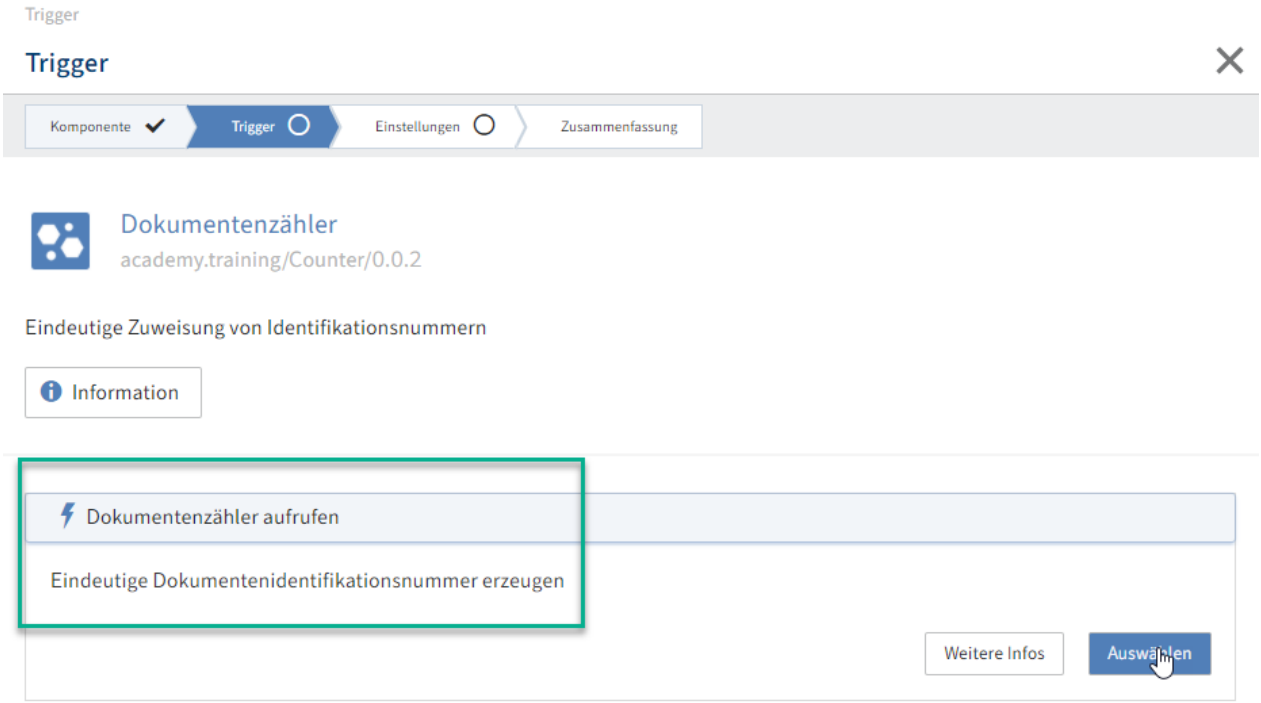
Definition der Komponentenklasse (hier CounterComponent):

```
@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName =
public class CounterComponent {
}
```

Trigger

Trigger werden zum Aufruf eines Flows verwendet.

Wird eine Komponente in einem Flow als erste Komponente eingesetzt, so stellt sie automatisch in der ELO Flows Administration, Trigger zur Verfügung.



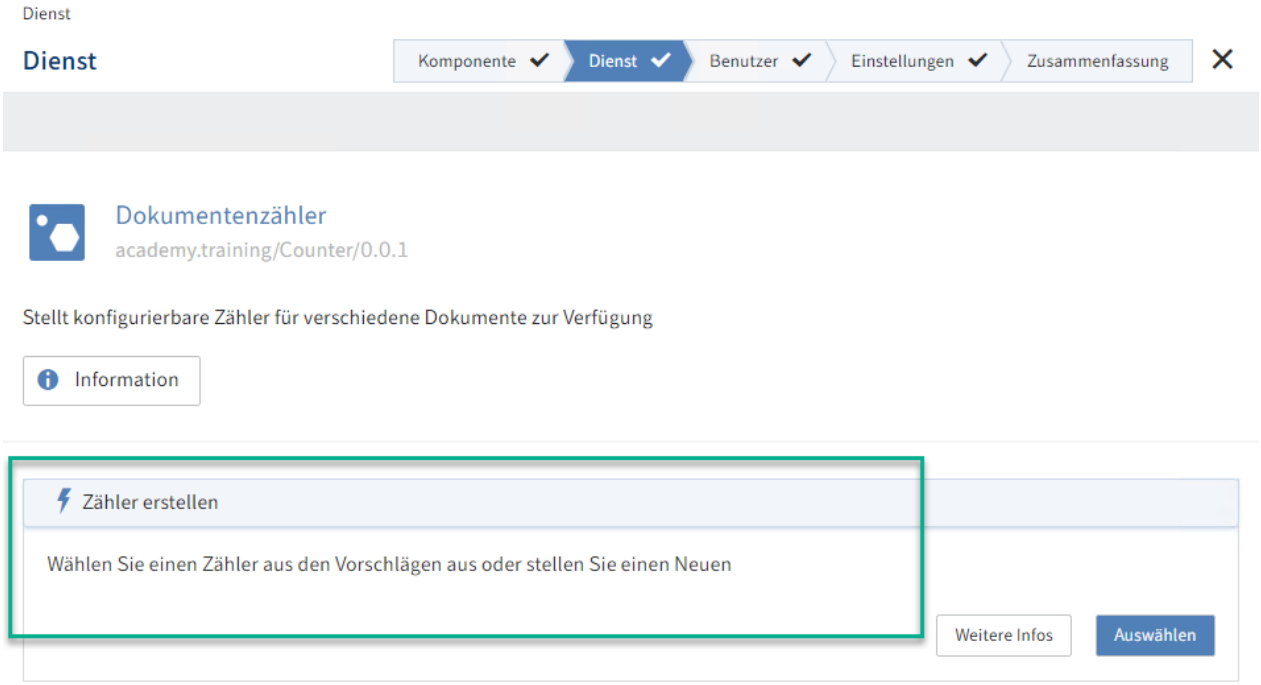
Beachten Sie

Trigger werden immer in der Komponentenklasse definiert.

```
@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName =
public class CounterComponent {

  @Trigger(displayName="Dokumentenzähler aufrufen")
  @WebHook(endpoint="createCounter")
  public CounterTriggerData triggerCounter(@Config CounterTriggerData triggerData) {
    //Implementierung
  }

  @Trigger(…)
  @Scheduled(…)
  public TriggerData otherTrigger(@Config TriggerData triggerData) {
  //Implementierung
  }
}
```

Dienst

Dienste stellen unterschiedliche Funktionalitäten in ELO Flows zur Verfügung. Wird eine Komponente nicht als erste Komponente in ELO Flows verwendet, dann stellt sie unterschiedliche Dienste zur Auswahl bereit. Stehen mehrere Dienste zur Auswahl, so können sie zusätzlich gruppiert werden.



> **Beachten Sie**
>
> Dienste werden in der Komponentenklasse definiert.

```
@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName =
public class CounterComponent {
  @Service(…)
  public ServiceOutputData createCounter(SerivceInputData ) {
    //Implementierung
  }
}
```

Eingabefelder

Die Eingabefelder dienen der Erfassung der Benutzerdaten. Die Felder können von verschiedenem Typ sein und zusätzlich Eingabevorschläge anbieten. Über die Eingabefelder können ebenfalls Daten aus vorhergehenden Komponenten über Schlüssel verwendet werden. Die Eingabefelder werden an einen Dienst `@Service` bzw. Trigger `@Trigger` übergeben, wo sie für Konfigurationsaufgaben im Ablauf des Flows genutzt werden.

Folgende Typen sind möglich:

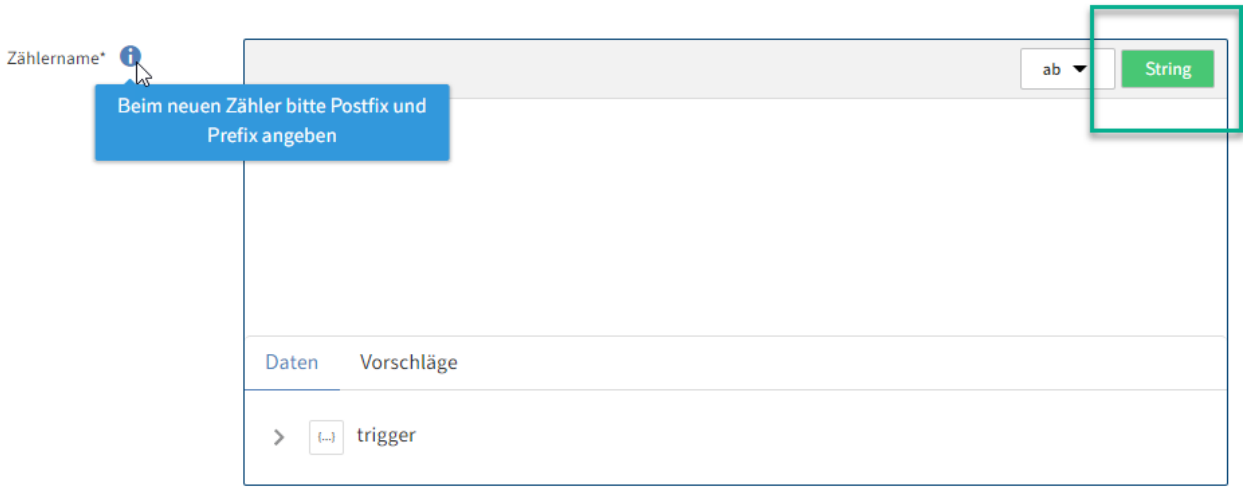
- String
-

Integer
- Boolean
- Object
- Array

Eingabefeld String (Text)

Die Eingabefelder für Werte unterschiedlichen Typs beinhalten zusätzlich den Konfigurationsbaum und einen JSONata-Editor.

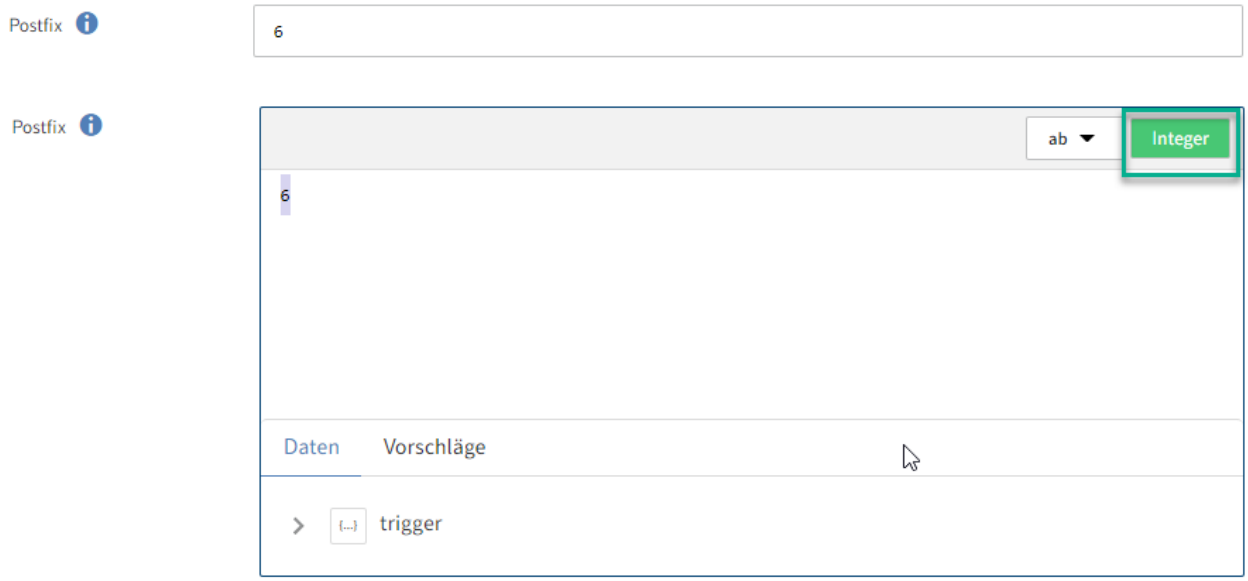
Um ein Eingabefeld umzusetzen wird eine Klasse deklariert, die Felder über die Annotation `@Property` bereitstellt. Diese Klasse wird als Parameter an eine Dienst- bzw. Trigger-Methode übergeben. Der Typ der Variablen (*String counterName* unter der Annotation `@Property`) bestimmt den Typ des Feldes im Editor.




```
CounterInput.java
public class CounterInput {
  @Property(displayName="Zählername" description="Beim neuen Zähler bitte Postfix und Prefix
  private String counterName; // Typ String im Eingabefeldeditor
  //Getter- und Settermethoden implementieren
}

CounterComponent.java
@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName =
public class CounterComponent {
  @Service(name="CreateCounter")
  public CounterOutput createCounter(CounterInput input) {}
}
```

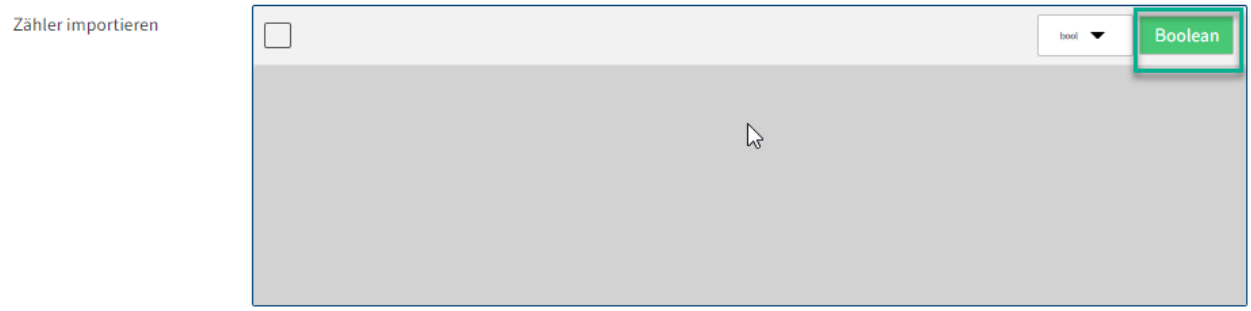
Eingabefeld Integer

Um ein Eingabefeld vom Typ Integer umzusetzen, wird analog zum Eingabefeld String vorgegangen. Der Typ der Variablen, in unserem Beispiel *postfix* wird vom Typ `int` deklariert.




```
CounterInput.java
public class CounterInput {
  @Property(displayName="Postfix")
  private int postfix; //Typ Integer im Eingabefeldeditor
  //Getter- und Settermethoden implementieren
}

CounterComponent.java
@Component(version = "0.0.1", namespace = "academy.training",
name = "Counter", displayName = "Dokumentenzähler",
description = "Eindeutige Zuweisung von Identifikationsnummern")
public class CounterComponent {
  @Service(name="CreateCounter")
  public CounterOutput createCounter(CounterInput input) {}
}
```

Eingabefeld Boolean




```
CounterInput.java
public class CounterInput {
  @Property(displayName="Zähler importieren")
  private boolean importCounter; //Typ Boolean im Eingabefeldeditor
  //Getter- und Settermethoden implementieren
}

CounterComponent.java
@Component(version = "0.0.1", namespace = "academy.training",
name = "Counter", displayName = "Dokumentenzähler",
description = "Eindeutige Zuweisung von Identifikationsnummern")
public class CounterComponent {
  @Service(name="CreateCounter")
  public CounterOutput createCounter(CounterInput input) {}
  }
}
```

Eingabefeld Object

Werden gleichzeitig mehrere Eingabezeilen (auch mit unterschiedlichen Typen) benötigt, so kann ein Eingabefeld vom Typ *Object* implementiert werden. Hier können mehrere Felder zu einem Objekt zusammengefasst werden.

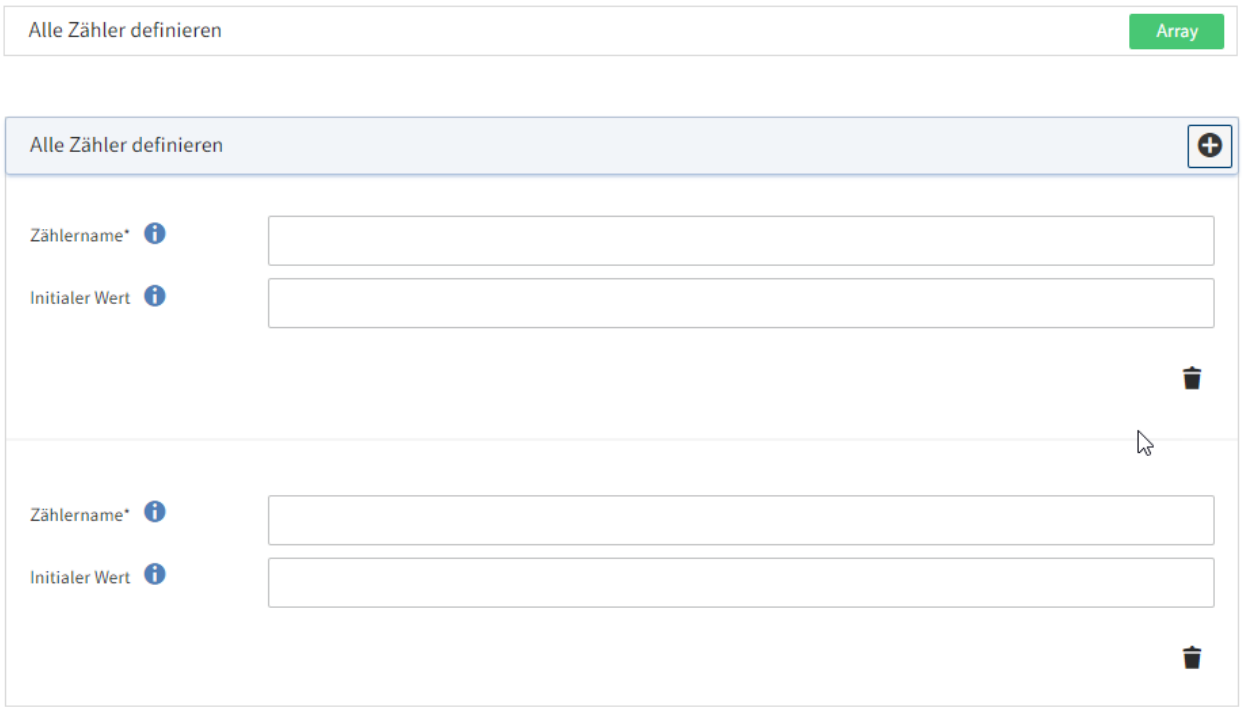
Das Beispiel zeigt die Umsetzung der *CounterObject* Klasse als innere Klasse.

```
public class CounterInput {
  @Property(displayName="Zähler definieren", description="Definieren Sie einen Zähler. Name u
  private CounterObject counterObj;//Typ CounterObject im Eingabefeldeditor
```

```
  class CounterObject {
    @Property(displayName="Initialer Wert", description="Der initiale Wert ist optional. Der
    @DisplayOptions(order = 2, suggestValue = true)
    private int value = 1;

    @Property(displayName="Zählername", description="Bitte tragen Sie Zählerbezeichnung ein"
    @DisplayOptions(order = 1)
    private String counterName;

    //Getter- und Settermethoden implementieren
  }
}

@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName =
public class CounterComponent {
  @Service(name="CreateCounter")
  public CounterOutput createCounter(CounterInput input) {}
}
```

Eingabefeld Array

Werden von einem Typ (String, Boolean, Integer oder Objekt) mehrere Felder benötigt, so können diese zu einem Array zusammengefasst werden. In dem unten aufgeführten Beispiel werden mehrere Objekteingabefelder gruppiert.




```
public class CounterInput {
  @Property(displayName="Zähler definieren", description="Definieren Sie einen Zähler mit Nam
  private CounterObject[] counterObjects;//Typ CounterObject[] im Eingabefeldeditor
```

```
  class CounterObject {
    //siehe Eingabefeld Object
  }
}
```

Pflichtfelder

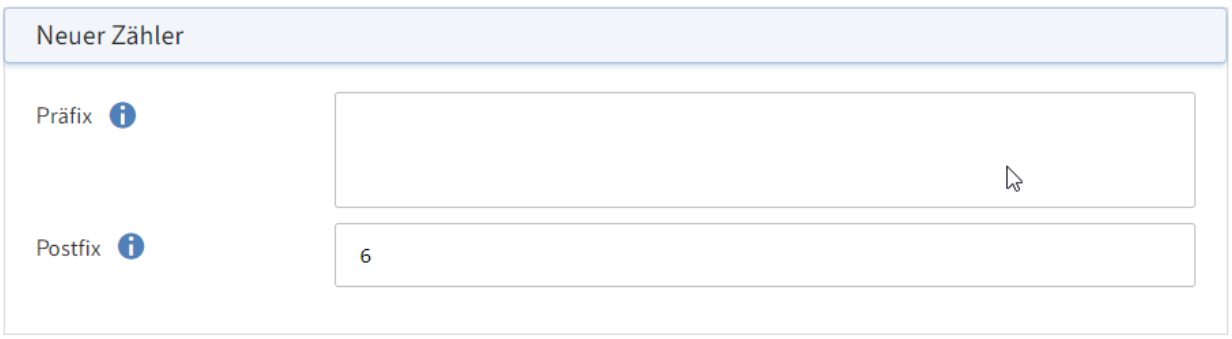
Alle Eingabefelder können als Pflichtfelder gekennzeichnet werden. Dabei wird der Bezeichner des Feldes mit einem Stern (*) markiert. Dadurch kann auch die gesamte Eingabemaske nicht verlassen werden, wenn die Eingaben fehlen.

Durch das Setzen des Attributes `required` auf `true` in der `Property` Annotation, wird programmtechnisch das Feld als Pflichtfeld markiert.

```
@Property(displayName="Zählername" description="Beim neuen Zähler bitte
Postfix und Prefix angeben", required=true)
private String counterName; // Typ String im Eingabefeldeditor
```

Reihenfolge, Vorbelegung der Felder

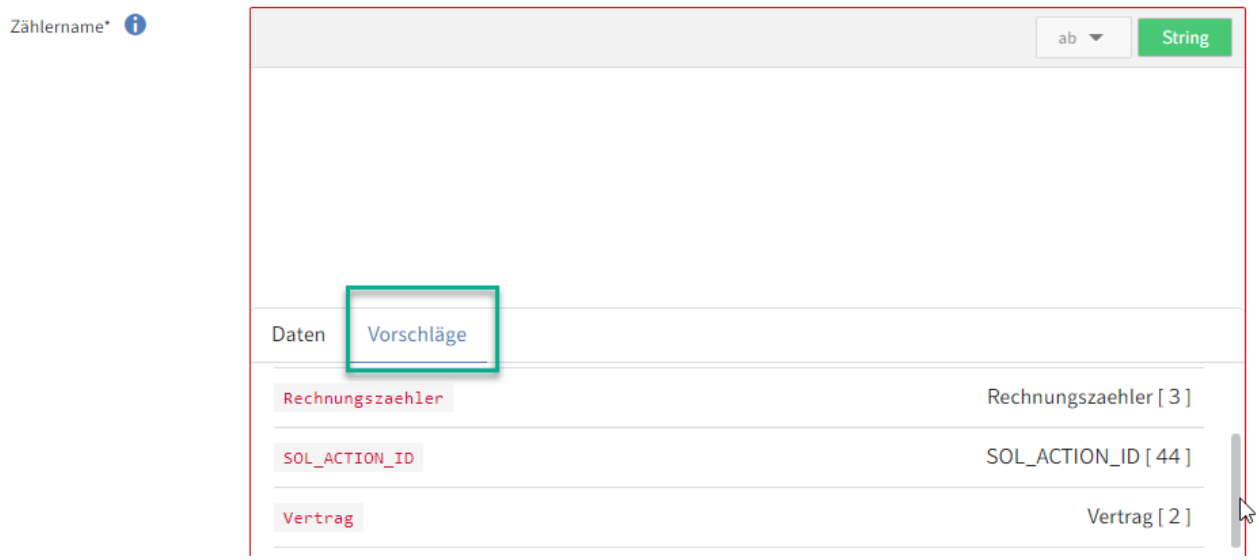
Über die Annotation `@DisplayOptions` können die Felder zusätzlich konfiguriert werden. Hier kann z. B. die Vorbelegung (suggestValue) bzw. die Reihenfolge (order) der Felder gesetzt werden.




```
@Property(displayName="Postfix")
@DisplayOptions(order=2, suggestValue=true)
private int postfix = 6;

@Property(displayName="Prefix")
@DisplayOptions(order=1, size=2)
private String prefix = "Rechnung";
```

Tab 'Vorschläge'

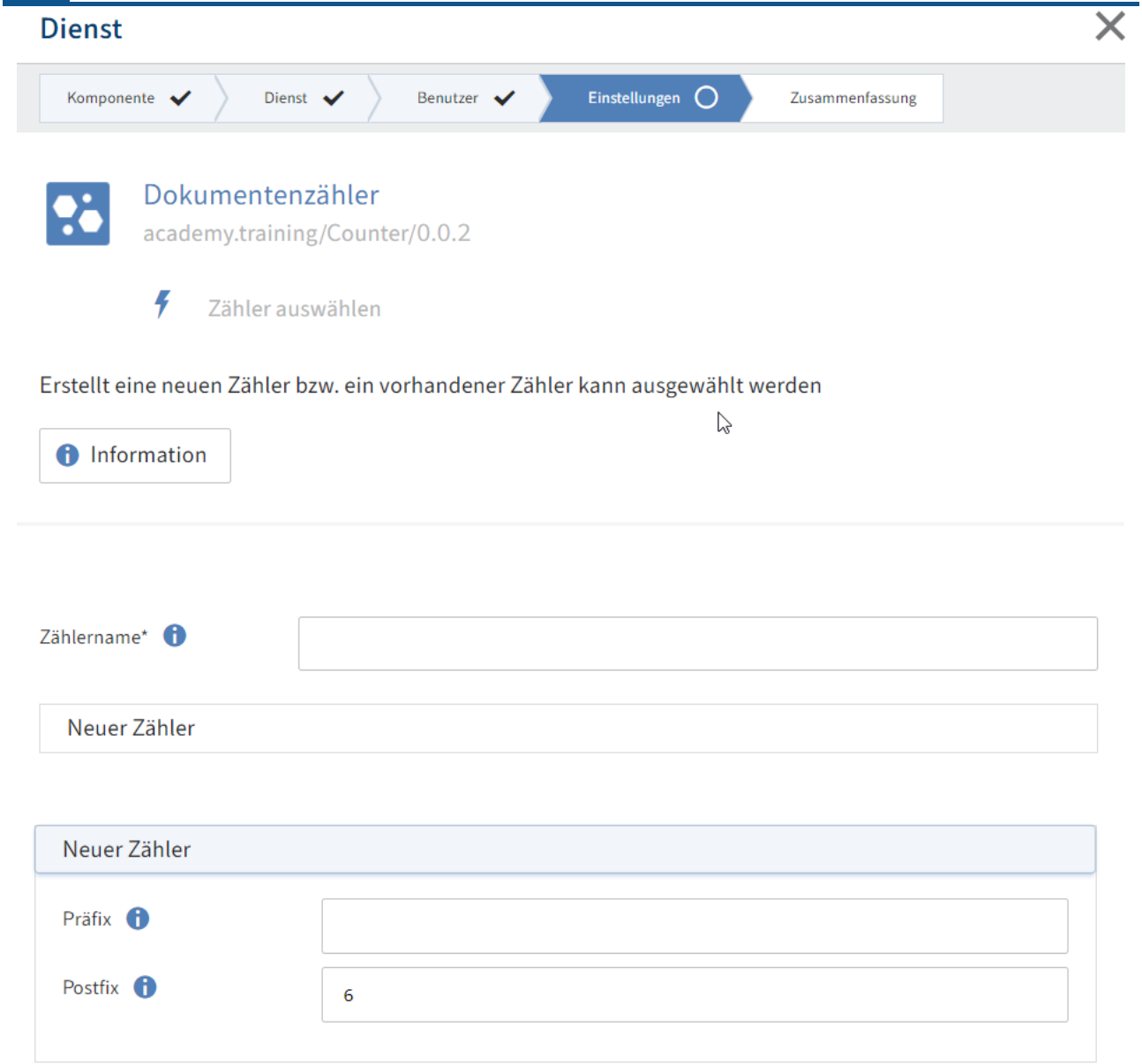
In allen Eingabefeldern können Vorschlaglisten über die Annotationen `@Lookup` und `@LookupProvider` gefüllt werden. Die Implementierung der Funktion mit der Annotation `@LookupProvider` muss in der Hauptkomponentenklasse `@Component` umgesetzt sein.




```
public class CounterInput {
  @Property(displayName = "Zählername", required=true)
  @Lookup("getCounters")
  private String counterName;
}

@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName =
public class CounterComponent {

  @LookupProvider("getCounters")
  public Map<String,String> getCounters() throws CounterException {
  HashMap<String,String> map = new HashMap<>();
    //Implementierung
    return map;
  }
}
```

Feldergruppen

Die Eingabefelder können zu Gruppen thematisch zusammengefasst werden. In der Abbildung wurden die Felder *Prefix* und *Postfix* in der Gruppe *Neuer Zähler* zusammengefasst.



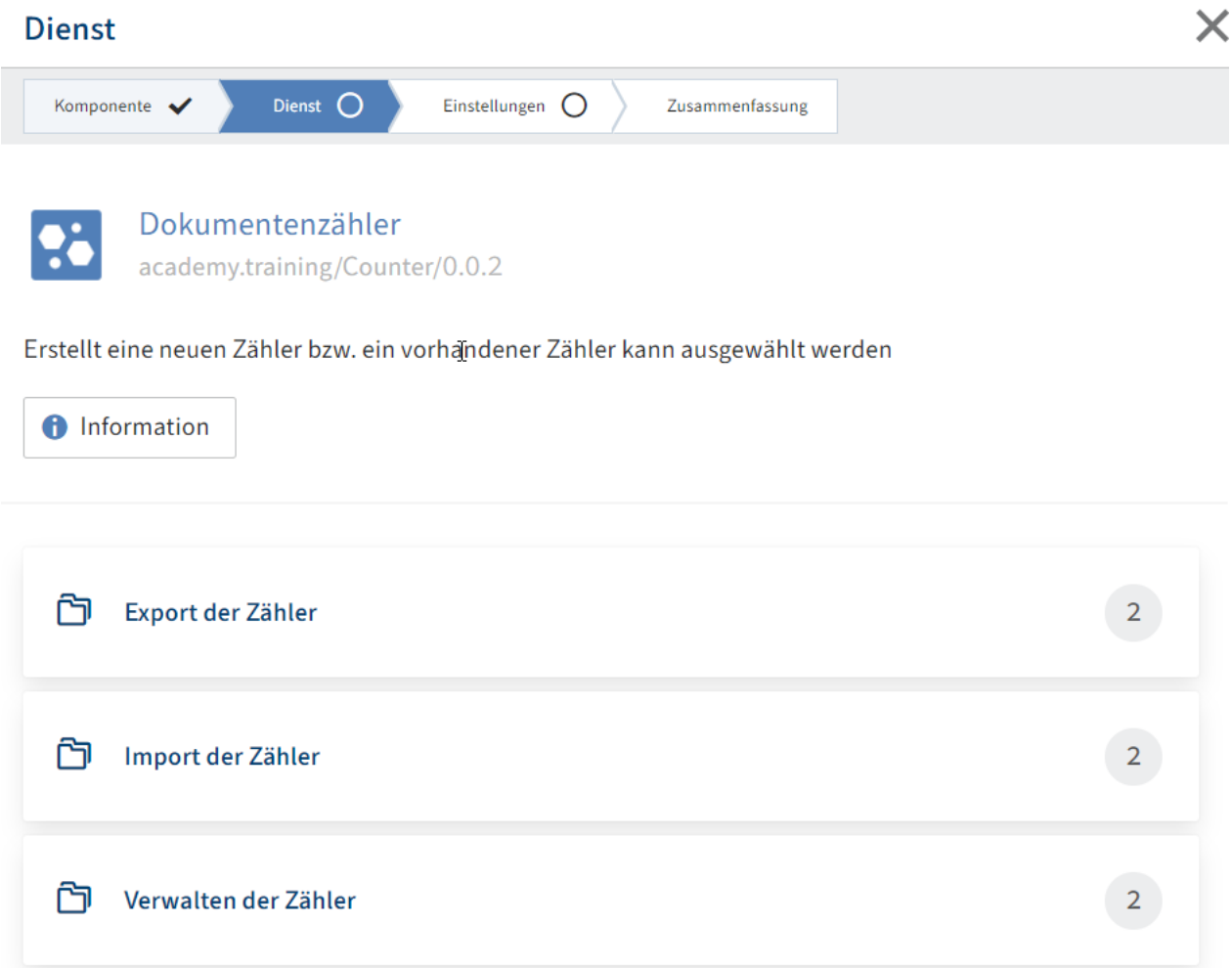
Um die Gruppierung von Eingabefelder zu erreichen, wird die Klassendeklaration um die Annotationen `@PropertyGroups` und `@PropertyGroup` erweitert. Die Felder, die zu einer Gruppe zusammengefasst werden, werden zusätzlich um die Annotation `@PropertyGroupRef` mit dem entsprechenden Gruppennamen erweitert.

```
@PropertyGroups(@PropertyGroup(displayName="Neuer Zähler", name="counterGroup"))
public class CounterInput {

  @Property(displayName = "Zählername", required = true, description = "Bitte wählen Sie eine
  @Lookup("getCounters")
  private String counterName;

  @Property(displayName = "Prefix", description = "Nur in Verbindung mit Zählername")
  @PropertyGroupRef("countergroup")
  @DisplayOptions(order = 1)
  private String prefix;
```

```
  @Property(displayName = "Postfix", description = "Nur in Verbindung mit Zählername")
  @PropertyGroupRef("countergroup")
  @DisplayOptions(order = 2, suggestValue = true)
  private int postfix = 6;
}
```

Dienstgruppen

Verschiedene Dienste, siehe auch die *ELO Objekte & Metadaten*-Komponente, können in Gruppen zusammengefasst werden.



Um die Gruppierung der Dienste zu erreichen, werden die Dienste in einzelnen Klassen implementiert, die um die Annotation `@ComponentServices` erweitert werden. Damit die Gruppen angezeigt werden, müssen mindestens zwei Gruppen definiert sein (d.h. zwei Klassen mit `@ComponentServices` und darin enthaltenen `@Services`). Die Annotation `@ComponentServices` muss immer über das Attribut `component` auf die Komponentenhauptklasse (hier `CounterComponent`) verweisen.

```
@Component(version = "0.0.1", namespace = "academy.training", name = "Counter", displayName =
public class CounterComponent {
}

@ComponentServices(component = CounterComponent.class, name = "CounterServices", displayName
```

```
public class CounterServices {

  @Service(name="CreateCounter")
  public CounterOutput createCounter(CounterInput input) {}

  @Service(name="ChooseCounter")
  public CounterOutput chooseCounter(CounterInput input) {}
}

@ComponentServices(component = CounterComponent.class, name = "CounterServices", displayName
public class CounterServicesImportGroup {

  @Service(name="ImportCounter")
  public CounterImportOutput importCounter(CounterImportInput input) {}

  @Service(name="FindImportCounter")
  public CounterImportOutput findImportCounter(CounterImportInput input) {}
}
```

Indexserververbindung

Die Verbindung zum ELO Indexserver kann automatisch über die Annotation `@Connection` und `@ConnectionRequired` hergestellt werden. Über die Annotation `@Connection` in der Variablen vom Typ *IXConnection* wird die Verbindung zum ELOix in einer Klasse deklariert und kann über die Annotation `@ConnectionRequired` in jeder Methode verwendet werden.

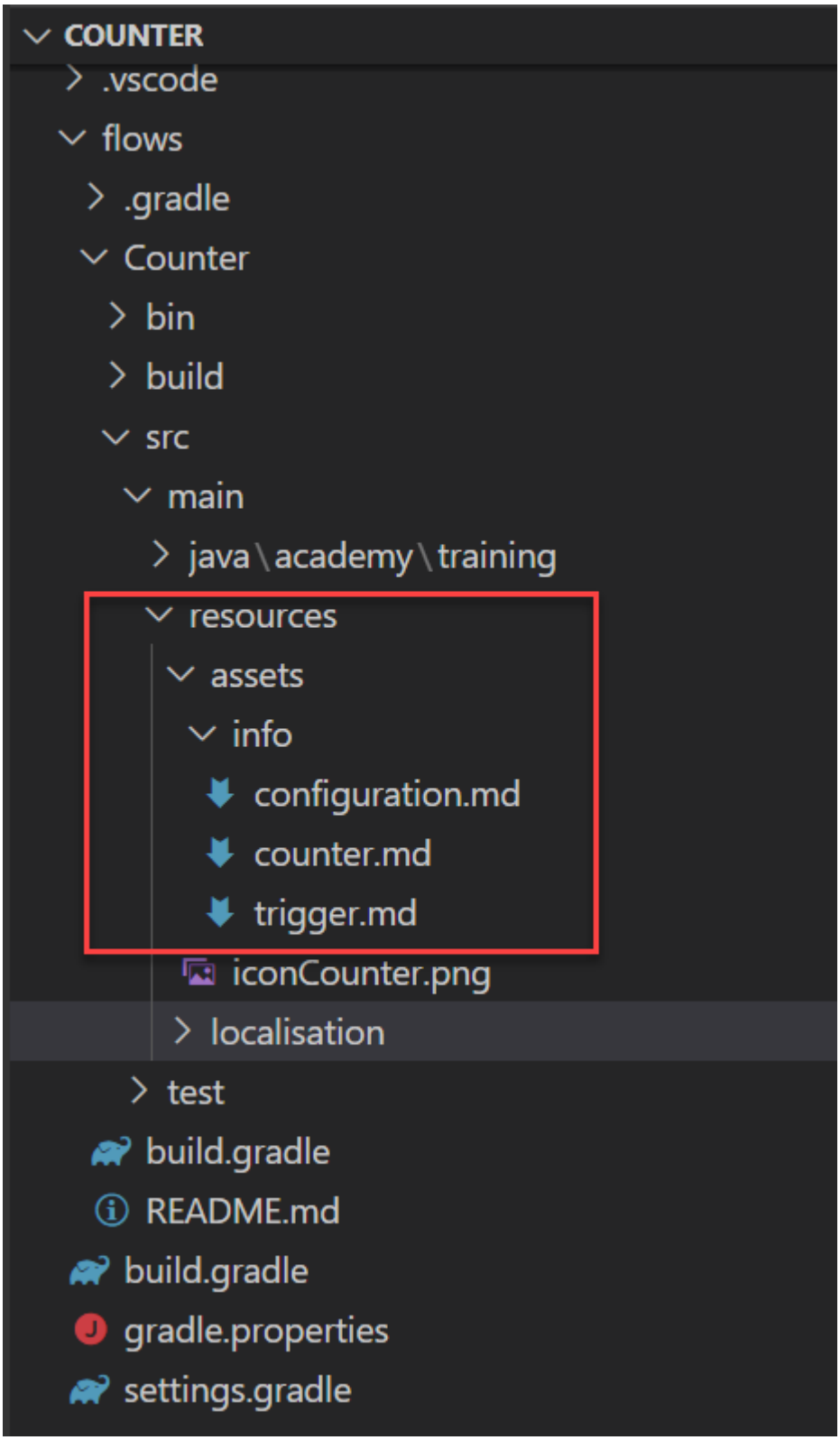
```
public class CounterComponent {

  @Connection
  IXConnection ixConnect;

  @LookupProvider("getCounters")
  @ConnectionRequired
  public Map<String,String> getCounters() {
    HashMap<String,String> map = new HashMap<>();
    //Implementierung
    return map;
  }
}
```

Komponenteninformation

Die Dokumentation kann über die Annotaion `@Guide` an verschiedenen Stellen, z. B. Dienst- oder Komponentenbeschreibung, eingebunden werden. Dafür ist zunächst in der entsprechenden Projektstruktur die relevante Markdown-Datei abzulegen, auf die später in der Implementierung der Lösung verwiesen wird.

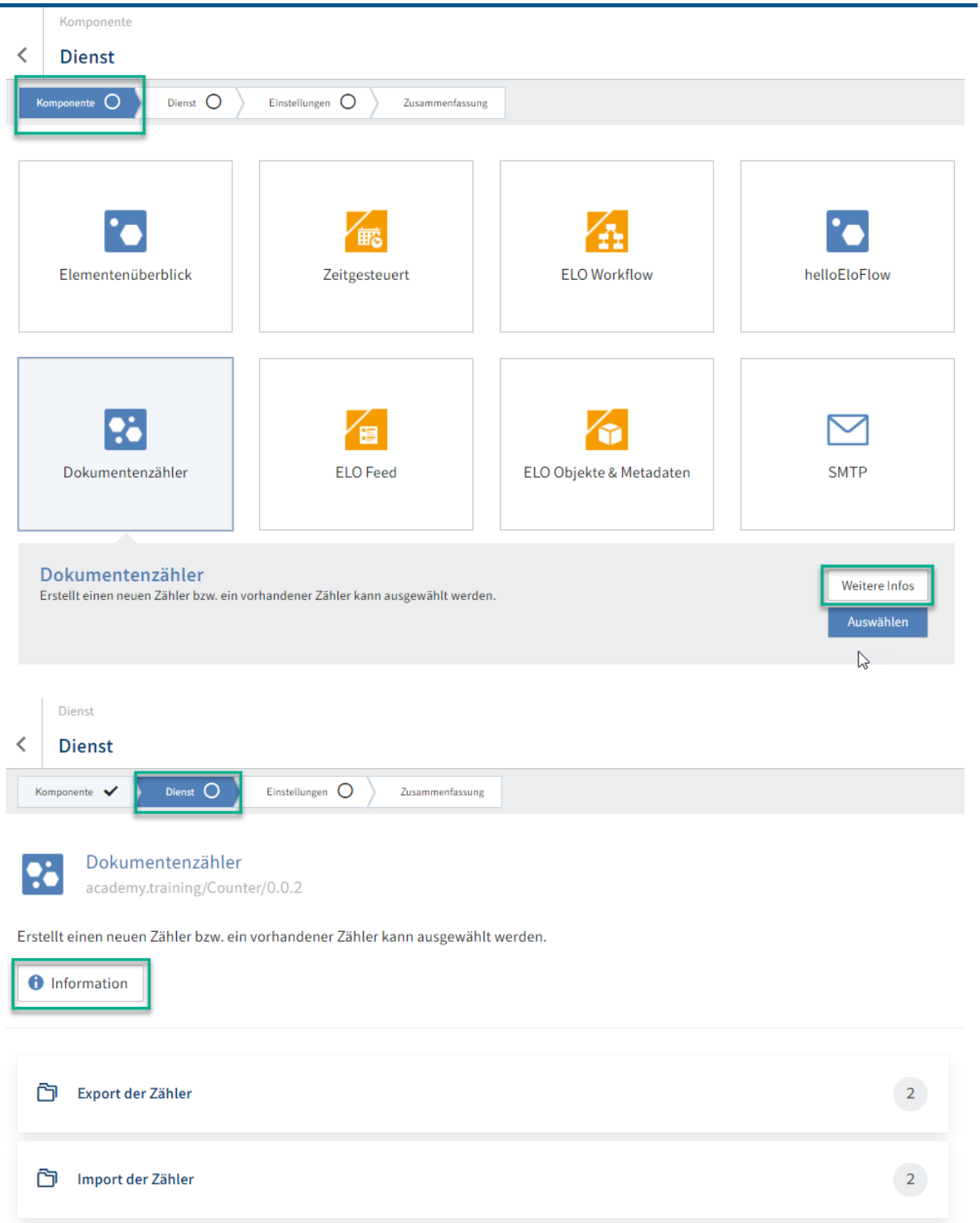
> **Beachten Sie**

Die Markdown-Dateien werden in der dafür vorgesehenen Projektstruktur (siehe Abbildung) abgelegt. Ansonsten kann die Dokumentation nicht gelesen und angezeigt werden.

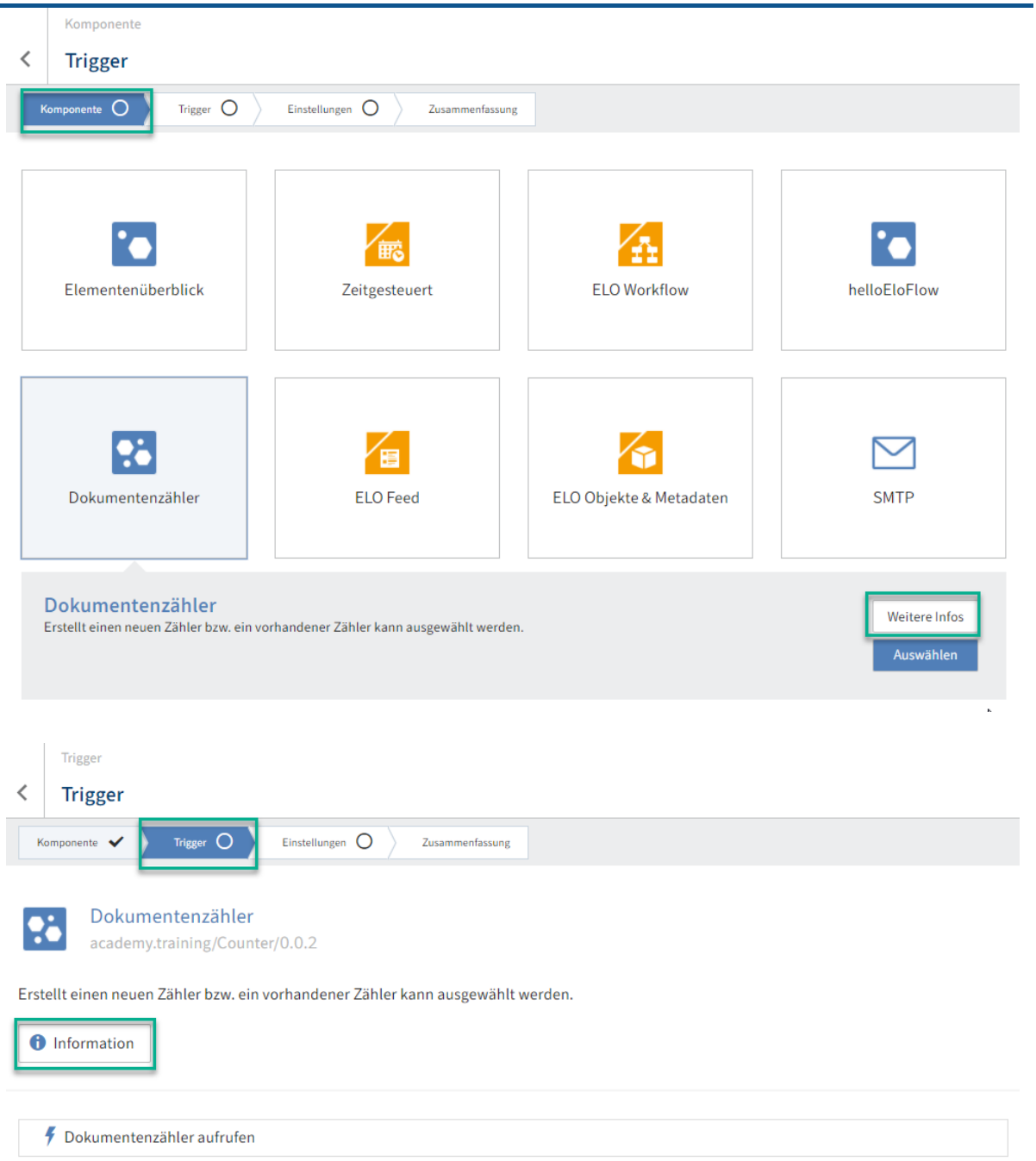


Folgende Kombinationsmöglichkeiten sind mit anderen Annotationen möglich:

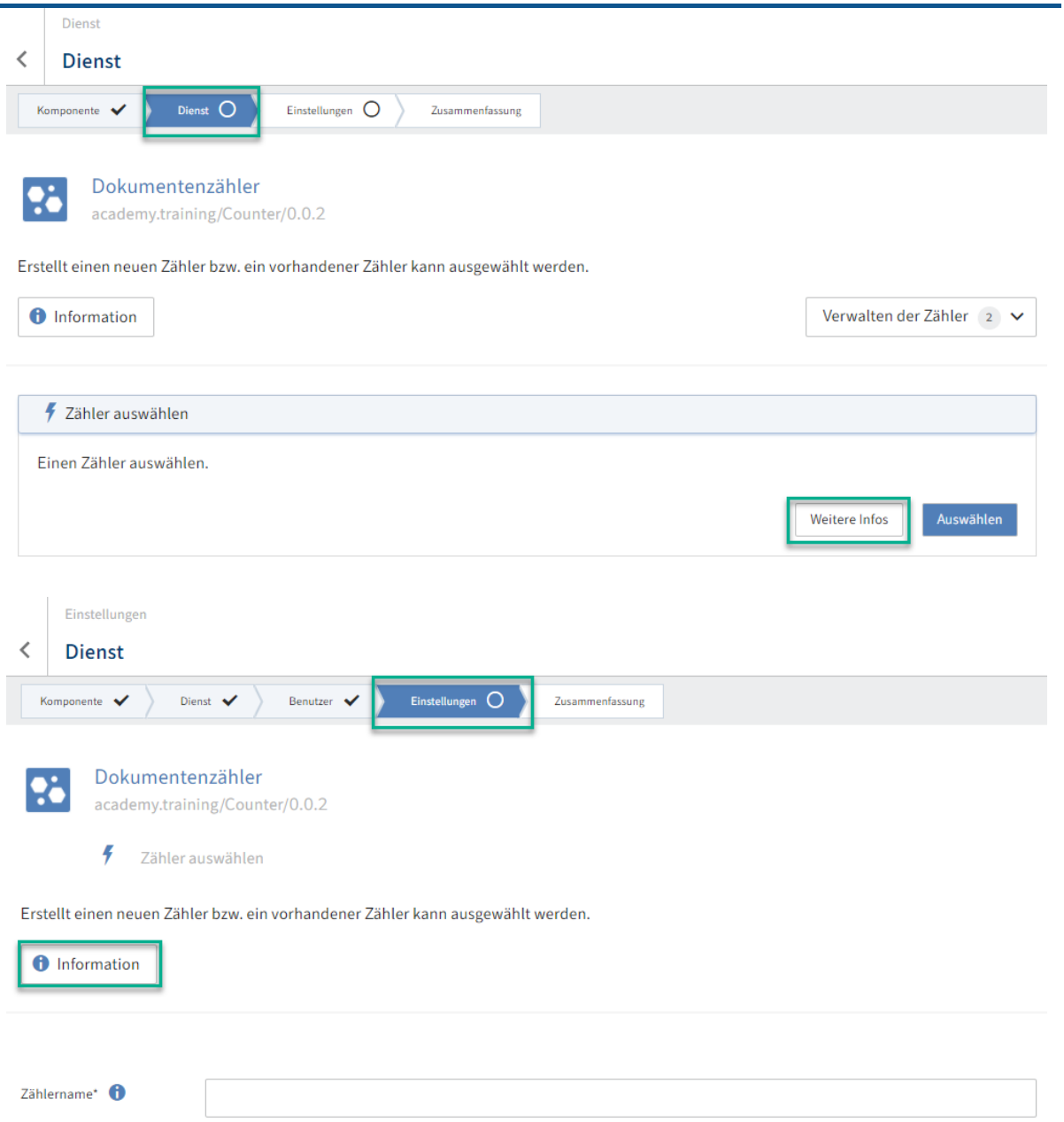
1. Informationen über eine Komponente (`@Component` und `@Guide`). Sie sind sowohl in der Dienste- als auch in Trigger-Auswahlansicht sichtbar.


Komponente
Dienst
Komponente
Dienst
Einstellungen
Zusammenfassung
Elementenüberblick
Zeitgesteuert
ELO Workflow
helloEloFlow
Dokumentenzähler
ELO Feed
ELO Objekte & Metadaten
SMTP
Dokumentenzähler
Erstellt einen neuen Zähler bzw. ein vorhandener Zähler kann ausgewählt werden.
Weitere Infos
Auswählen

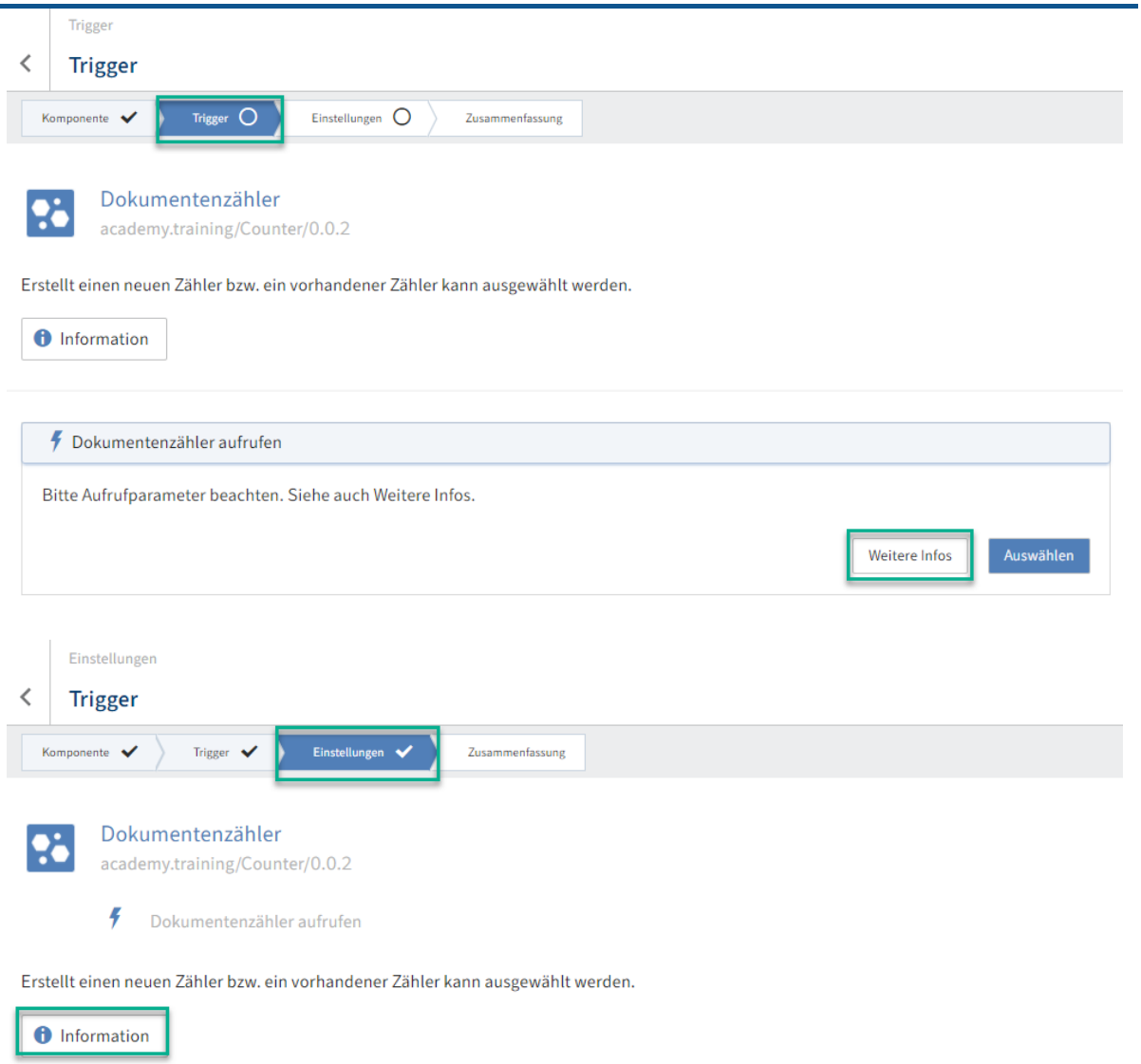
Dienst
Dienst
Komponente
Dienst
Einstellungen
Zusammenfassung
Dokumentenzähler
academy.training/Counter/0.0.2
Erstellt einen neuen Zähler bzw. ein vorhandener Zähler kann ausgewählt werden.
Information
Export der Zähler
2
Import der Zähler
2



2. Informationen über einen Dienst (`@Service` und `@Guide`).



3. Informationen über einen Trigger (`@Trigger` und `@Guide`)


Trigger
Trigger
Komponente
Trigger
Einstellungen
Zusammenfassung
Dokumentenzähler
academy.training/Counter/0.0.2
Erstellt einen neuen Zähler bzw. ein vorhandener Zähler kann ausgewählt werden.
Information
Dokumentenzähler aufrufen
Bitte Aufrufparameter beachten. Siehe auch Weitere Infos.
Weitere Infos
Auswählen

Einstellungen
Trigger
Komponente
Trigger
Einstellungen
Zusammenfassung
Dokumentenzähler
academy.training/Counter/0.0.2
Dokumentenzähler aufrufen
Erstellt einen neuen Zähler bzw. ein vorhandener Zähler kann ausgewählt werden.
Information

Übersicht Annotationen

Ein Überblick über die Annotationen, die uns bei der Umsetzung zur Verfügung stehen.

@Component

```
package com.elo.flows.api.components.annotations;

public abstract @interface Component {
    public abstract java.lang.String namespace();
    public abstract java.lang.String name();
    public abstract java.lang.String version();
    public abstract java.lang.String displayName();
    public abstract java.lang.String iconUri() default "/assets/icon.png";
    public abstract java.lang.String description() default "";
}
```

Name	Beschreibung
namespace	Namensraum (Paketierung im Projekt).
name	Name der Komponente.
version	Version der aktuellen Version.
displayName	Anzeigename der in der Flows Administration angezeigt wird.
iconUri	Pfad zum Icon der Komponente, Standard /assets/icon.png.
description	Kurze Beschreibung der Komponente, die nach dem Markieren der Komponente angezeigt wird. Weitere Informationen werden über die Annotation `@Guide` in der Komponente hinterlegt.

@Service

Die Dienste, die in einer Komponente zur Verfügung gestellt werden.

```
package com.elo.flows.api.components.annotations;

public abstract @interface Service {
    public abstract java.lang.String name() default "";
    public abstract java.lang.String displayName() default "";
    public abstract java.lang.String description() default "";
}
```

Name	Beschreibung
name	Name des Dienstes.
displayName	Anzeigename der in der Flows Administration angezeigt wird.
description	Kurze Beschreibung des Dienstes, die nach dem Markieren des Dienstes angezeigt wird. Weitere Informationen werden über die Annotation `@Guide` im Dienst hinterlegt.

@Trigger, @WebHook

Trigger über den ein Flow gestartet werden kann.

```
package com.elo.flows.api.components.annotations;

public abstract @interface Trigger {
    public abstract java.lang.String name() default "";
    public abstract java.lang.String displayName() default "";
    public abstract java.lang.String description() default "";
}

package com.elo.flows.api.components.annotations;

    public abstract @interface WebHook {
    public abstract java.lang.String endpoint() default "";
}
```

Name	Beschreibung @Trigger
name	Name des Triggers.
displayName	Anzeigename der in der Flows Administration angezeigt wird.
description	Kurze Beschreibung des Triggers, der nach dem Markieren des Triggers angezeigt wird. Weitere Informationen werden über die Annotation `@Guide` im Trigger hinterlegt.

Name	Beschreibung @WebHook
endpoint	Bezeichner für den Aufruf.

@Config

Wird in einem Trigger (Trigger-Methode) als Parameter verwendet. Zeigt an, dass im Trigger im Tab *Einstellungen* Konfigurationen vorgenommen werden können.

```
package com.elo.flows.api.components.annotations;

public abstract @interface Config {
}
```

@Synchron

Kennzeichnet einen synchronen Trigger.

```
package com.elo.flows.api.components.annotations;
public abstract @interface Synchron {

  public abstract  java.lang.Class<? extends java.lang.Object>[] resultClasses() default {};
}
```

@Guide

Ermöglicht das Anbinden von Informationsdateien im Markdown-Format.

```
package com.elo.flows.api.components.annotations;

public abstract @interface Guide {
    public abstract java.lang.String value();
}
```

@Property

Feld im Tab *Einstellungen*.

```
package com.elo.flows.api.schema.annotations;

public abstract @interface Property {
    public abstract java.lang.String displayName() default "";
    public abstract java.lang.String description() default "";
    public abstract boolean required() default false;
}
```

Name	Beschreibung
displayName	Anzeigename der in der Flows Administration angezeigt wird.
description	Kurze Beschreibung des Eingabefeldes, die hinter dem (i) als Tooltip angezeigt wird.
required	Gibt an, ob es sich um ein Pflichtfeld handelt.

@DisplayOptions

Weiter Konfigurationsmöglichkeiten der Eingabefelder.

```
package com.elo.flows.api.schema.annotations;

public abstract @interface DisplayOptions {
    public abstract int size() default (int) 1;
    public abstract int order() default (int) 2147483647;
    public abstract boolean hidden() default false;
    public abstract boolean suggestValue() default false;
}
```

Name	Beschreibung
size	Größe des Eingabefeldes.
order	Bei mehreren Eingabefeldern in einer Gruppe, kann die Reihenfolge beim Anzeigen festgelegt werden.
hidden	Ein Feld kann als versteckt markiert werden.

Name	Beschreibung
suggestValue	Der Wert der @Property Variablen wird automatisch im Eingabefeld übernommen.

@Lookup

Vorschläge Tab im Eingabefeld. Wird mit der Annotation `@Property` verwendet. Weiter muss sichergestellt sein, dass über die Annotation `@LookupProvider` eine Methode implementiert wird, die den Inhalt für die Vorschläge bereitstellt.

```
package com.elo.flows.api.schema.annotations;

public abstract @interface Lookup {
    public abstract java.lang.String value();
}
```

@LookupProvider

Vorschläge im Eingabefeld. Wird in Verbindung mit der Annotation `@Lookup` verwendet.

```
package com.elo.flows.api.components.annotations;

public abstract @interface LookupProvider {
    public abstract java.lang.String value();
}
```

@Connection

Verbindung zum ELO Indexserver.

```
package com.elo.flows.api.components.annotations;

public abstract @interface Connection {
    public abstract java.lang.Class<?> provider() default java.lang.Void;
}
```

@ConnectionRequired

Wird mit `@Connection` benutzt, wenn z. B. in einer Methode die Verbindung zum ELO Indexserver genutzt wird.

```
package com.elo.flows.api.components.annotations;
    public abstract @interface ConnectionRequired {
}
```

@PropertyGroups, PropertyGroup, PropertyGroupRef

Die Annotationen werden benutzt, um im Tab *Dienste* die unterschiedlichen Dienste in Gruppen zusammenzufassen.

```
package com.elo.flows.api.schema.annotations;

public abstract @interface PropertyGroups {
    public abstract com.elo.flows.api.schema.annotations.PropertyGroup[] value();
}

package com.elo.flows.api.schema.annotations;

public abstract @interface PropertyGroup {
    public abstract java.lang.String displayName();
    public abstract java.lang.String name();
    public abstract boolean collapsed() default false;
    public abstract java.lang.String description() default "";
    public abstract int order() default (int) 2147483647;
}
```

Name	Beschreibung
displayName	Anzeigename der in der Flows Administration angezeigt wird.
name	Name der Gruppe.
collapsed	Die Gruppe kann beim Auswählen geöffnet sein.
description	Kurze Beschreibung der Gruppe. Wird hinter dem Gruppennamen angezeigt, wenn die Gruppe geöffnet ist.
order	Bei mehreren Einstellungen in der Gruppe, kann die Reihenfolge festgelegt werden.

```
package com.elo.flows.api.schema.annotations;

public abstract @interface PropertyGroupRef {
    public abstract java.lang.String value();
}
```