ELO Flows Development

Anhang

Inhaltsverzeichnis

Weitere Informationen

Implementierung Zähler

Wie bereits in den Anforderungen der Aufgabe festgehalten, wollen wir eine Komponente entwickeln, die einen Dokumentenzähler anbietet. Der Benutzer soll eine Möglichkeit zur Auswahl eines bereits vorhandenen Zählers erhalten, aber auch einen neuen Zähler anlegen können. Bei der Erstellung eines neuen Zählers soll die Eingabe eines Pre- und Postfixes erlaubt sein.

Die Anpassungen, die wir bereits in unserem Projekt durchgeführt haben, erlauben uns bereits eine Komponente zu erstellen, die eine Verbindung zum ELO Indexserver automatisch herstellt.

Die Bereitstellung der Komponente wird durch die Verwendung der Annotation `@Component`, direkt über dem Klassennamen im Code, umgesetzt.

Eine Verbindung zum ELO Indexserver wird über die Annotation `@Connection`, in unserem Fall über der Deklaration unserer Instanzvariablen `ixConnect`, automatisch hergestellt.

```
@Component(version = "0.0.1", namespace = "academy.training",
name = "Counter", displayName = "Counter.display.name")
public class CounterComponent {
  @Connection
  IXConnection ixConnect;
}
```

Folgende Implementierungsschritte werden wir jetzt gemeinsam umsetzen:

Logger für die Log-Ausgaben

Beachten Sie, dass die richtigen Importpakete ausgewählt werden. VS Code bietet Ihnen mehrere Importmöglichkeiten für die Klasse *Logger* an (*import org.slf4j.Logger*, *org.slf4j.LoggerFactory*)

```
private static final Logger LOG = LoggerFactory.getLogger(CounterComponent.class);
```

Dienstmethode für den Zähler (Tab 'Dienst')

Hierzu benötigen wir zuerst eine Klasse, die den Zählerwert über einen Schlüssel an die nachfolgende Komponenten im Flow weitergibt. Diese Klasse wird in unserer Projektstruktur unter *model* abgelegt.

```
package academy.training.model;
public class CounterOutput {
  private String counterValue;
  public String getCounterValue() {
    return counterValue;
```

```
  }
  public void setCounterValue(String counterValue) {
    this.counterValue = counterValue;
  }
}
```

Unsere Dienstimplementierung setzen wir in unserer Klasse CounerComponent.java mithilfe der Annotation `@Service` und `@ConnectionRequired` um. Damit wird sichergestellt, dass der Dienst in der Administration ausgewählt werden kann und der Dienst über eine Verbindung zum ELO Indexserver verfügt. Die Implementierung werden wir im weiteren Verlauf unserer Schulung sukzessive erweitern.

```
@Service(displayName = "Zähler auswählen")
@ConnectionRequired
public CounterOutput createCounter() {
  if (LOG.isDebugEnabled()) {
    LOG.debug("createCounter start");
  }
  CounterOutput counterOutput = new CounterOutput();
  String counterValue = "undefined";
  //TODO getCounterValue
  return counterOutput;
}
```

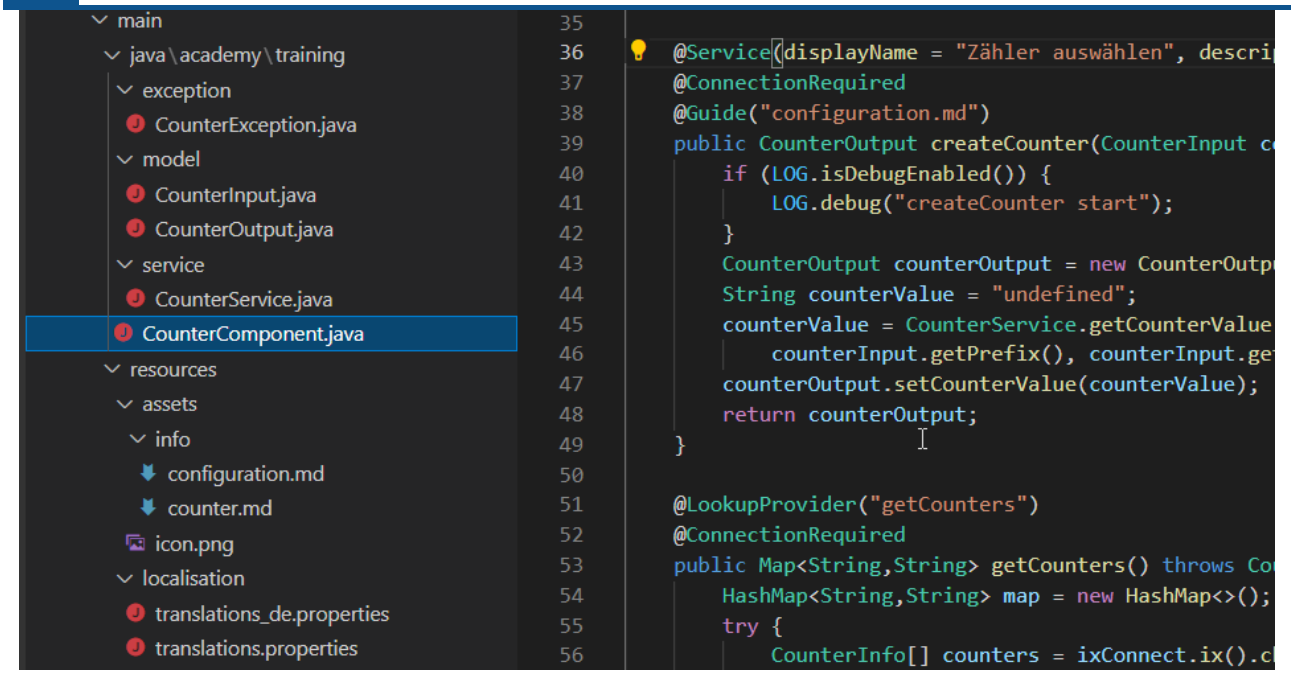
Eingabefelder für Zählername, Pre- und Postfix (Tab 'Einstellungen')

Hierzu benötigen wir ebenfalls eine neue Klasse *CounterInput.java*, die in der Projektstruktur unter *model* abgelegt wird. In dieser Klasse werden wir über die Annotationen die Elemente für die Zählerauswahl (Tab *Einstellungen*) zuweisen. Über die Annotation `@Property` werden die einzelnen Eingabefelder definiert, die über `@PropertyGroups` und `@PropertyGroup` gruppiert werden können. In den Gruppen kann die Reihenfolge über `@DisplayOptions` festgelegt werden. Wir wollen auch die Vorschlagliste mit bereits vorhandenen Zählern füllen. Hierzu verwenden wir die Annotation `@Lookup` und implementieren eigene Logik auf Basis der ELOix API.

> **Achtung**
>
> Logik für die Vorschläge `@Lookup` muss unbedingt in der Komponentenklasse `@Component` programmiert werden. Dort wird sie über `@LookupProvider` zur Verfügung gestellt.

Die neue Klasse *CounterInput.java* muss weiterhin in der Komponentenklasse in der Servicemethode `@Service` als Objekt übergeben werden. Zum Schluss implementieren wir die Logik für die Zählererstellung in der CounterService.java in der *service* Projektstruktur.

Mögliche Umsetzung:



CounterException.java

```
public class CounterException extends Exception {
  public final static String COUNTERERROR = "Fehler in der Zählererstellung";
  public CounterException() {
    super(CounterException.COUNTERERROR);
  }
  public CounterException(String errorMessage) {
    super(errorMessage);
  }
}
```

CounterInput.java

```
public class CounterException extends Exception {

package academy.training.model;
import com.elo.flows.api.components.annotations.Guide;
import com.elo.flows.api.schema.annotations.DisplayOptions;
import com.elo.flows.api.schema.annotations.Lookup;
import com.elo.flows.api.schema.annotations.Property;
import com.elo.flows.api.schema.annotations.PropertyGroup;
import com.elo.flows.api.schema.annotations.PropertyGroupRef;
import com.elo.flows.api.schema.annotations.PropertyGroups;

@Guide("configuration.md")
@PropertyGroups({@PropertyGroup(displayName ="Neuer Zähler", name="groupNewCounter")})
public class CounterInput{

  @Property(displayName = "Zählername", required=true, description="Beim neuen Zähler bitte P
```

```
  @Lookup("getCounters")
  private String counterName;

  @Property(displayName ="Prefix", description="Bitte den Zählername oben angeben")
  @PropertyGroupRef("groupNewCounter")
  @DisplayOptions(order = 1)
  private String prefix;

  @Property(displayName ="Postfix", description="Bitte auch den Zählername oben angeben")
  @PropertyGroupRef("groupNewCounter")
  @DisplayOptions(order = 2, suggestValue = true)
  private int postfix = 6;

  public String getCounterName() {
    return counterName;
  }

  public void setCounterName(String counterName) {
    this.counterName = counterName;
  }

  public String getPrefix() {
    return prefix;
  }

  public void setPrefix(String prefix) {
    this.prefix = prefix;
  }

  public int getPostfix() {
    return postfix;
  }

  public void setPostfix(int postfix) {
    this.postfix = postfix;
  }
}
```

CounterComponent.java

```
package academy.training;
import de.elo.ix.client.*;

import java.util.HashMap;
import java.util.Map;

import com.elo.flows.api.components.annotations.*;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import academy.training.exception.CounterException;
import academy.training.model.CounterInput;
import academy.training.model.CounterOutput;
import academy.training.service.CounterService;

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

  private static final Logger LOG = LoggerFactory.getLogger(CounterComponent.class);

  @Connection
  IXConnection ixConnect;

  @Service(displayName = "Zähler auswählen", description = "Wählen Sie einen Zähler für Ihr D
  @ConnectionRequired
  @Guide("configuration.md")
  public CounterOutput createCounter(CounterInput counterInput) {
    if (LOG.isDebugEnabled()) {
      LOG.debug("createCounter start");
    }
    CounterOutput counterOutput = new CounterOutput();
    String counterValue = "undefined";
    counterValue = CounterService.getCounterValue(ixConnect, counterInputgetCounterName(),
    counterInput.getPrefix(), counterInput.getPostfix());
    counterOutput.setCounterValue(counterValue);
    return counterOutput;
  }

  @LookupProvider("getCounters")
  @ConnectionRequired
  public Map<String,String> getCounters() throws CounterException {
    HashMap<String,String> map = new HashMap<>();
    try {
      CounterInfo[] counters = ixConnect.ix().checkoutCounters(null, true, LockC.NO);
      for (CounterInfo counterInfo : counters) {
        map.put(counterInfo.getName(), counterInfo.getName());
      }
    } catch (Exception e) {
      throw new CounterException(e.getMessage());
    }
    return map;
  }
}
```

Logik für den Zähler

CounterService.java (wird während der Schulung zur Verfügung gestellt)

Beinhaltet die Implementierung der Logik des Zählers und kann sehr individuell implementiert werden.

Weitere mögliche Erweiterungen

Trigger der Komponente (REST)

Analog zum `@Service` wird eine Trigger-Methode in der `@Component` Komponentenklasse implementiert. Die dazu notwendigen Annotationen sind `@Trigger` und `@WebHook`. Auch hier werden Input- und Outputklassen implementiert.

Testframework JUnit einbinden

In der Konfigurationsdatei *build.gradle* folgende Einträge hinzufügen.

> **Achtung**
>
> Bitte beachten Sie, dass Sie die Einträge in der richtigen *build.gradle* ergänzen.

Eigene Bibliotheken einbinden

Zusätzliche Bibliotheken werden über zentrale Repositories bereitgestellt und können mit Gradle eingebunden werden.