Feature Flags mit Togglz

Christian Kaltepoth

ingenit
wir leben IT

Über mich

- Christian Kaltepoth
- Senior Software Developer
- Fokus: JEE, Web, JSF, CDI, ...
- Open Source Developer

ingenit GmbH & Co. KG

- Gegründet 1997
- Softwareprojekte Java & .NET
- Fokus: Webanwendungen & BI
- Schulung und Beratung
- 123domain.eu Domainmanagement

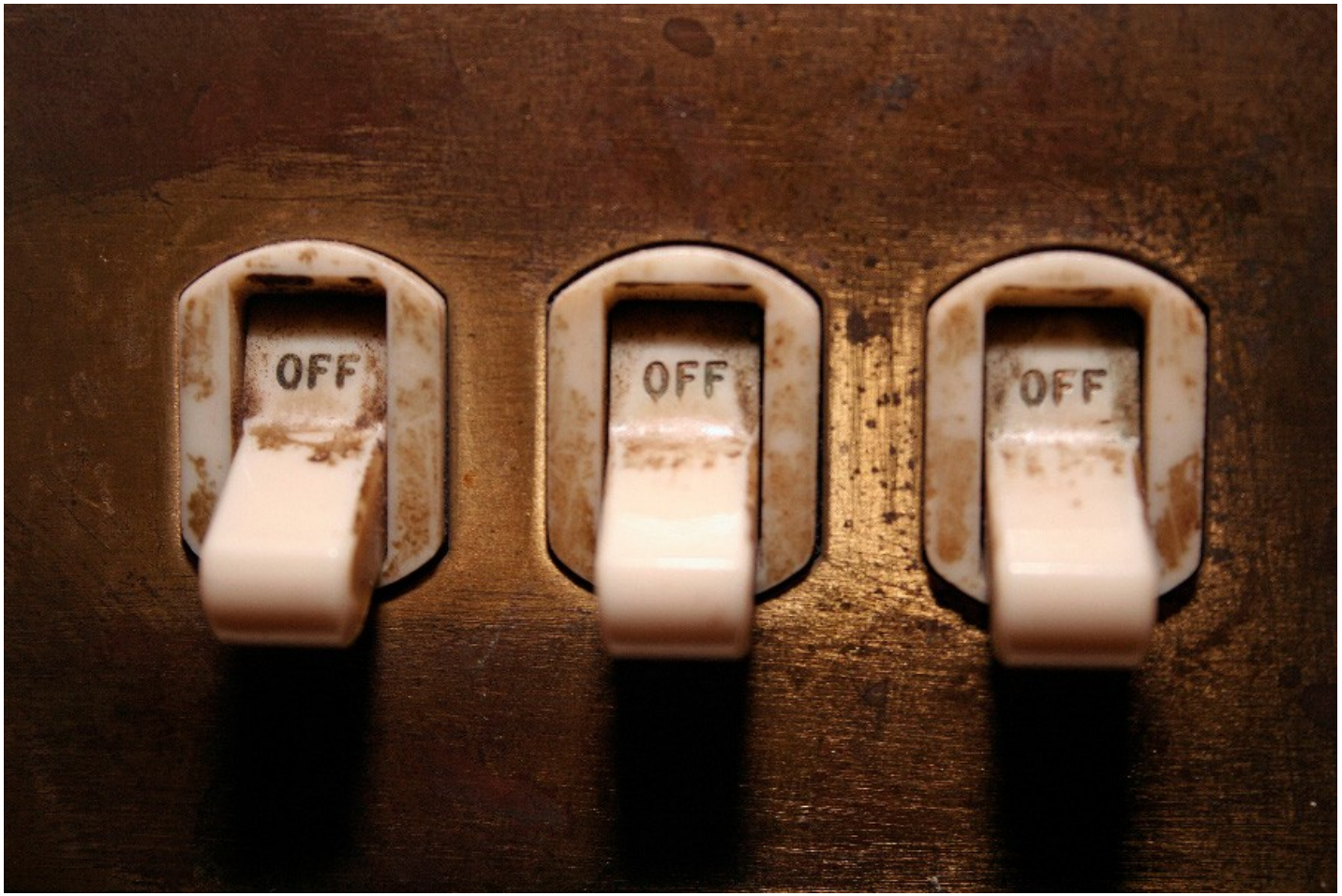
ingenit
wir leben IT

Continuous Delivery

Continuous Delivery Patterns

- Branching vermeiden / Branch by Abstraction
- Continuous Integration
- Blue / Green Deployment
- Continuous Deployment
- Dark Launching / Dark Testing
- Feature Toggles

Feature Toggles



Pending Features

Erprobung in Produktionssystem

Im Notfall

Feature Flags for the Java plattform

Entwicklungsziele

- Einfache Nutzung
- Geringer Integrationsaufwand
- Integration in die Anwendungsarchitektur
- Unterstützung von Security Frameworks

Was ist ein Feature?

- Neue Funktion
- Alternative Implementierung
- Optimierung
- ...

Installation

- Webseite: http://www.togglz.org/
- ZIP-Archiv / Maven Central
- Modular aufgebaut

Schritt 1:

Feature-Definition

Feature Definition

```
public enum MyFeatures implements Feature {

  @Label("First Feature")
  FEATURE_ONE,

  @EnabledByDefault
  @Label("Second Feature")
  FEATURE_TWO;

  @Override
  public boolean isActive() {
    return FeatureContext.getFeatureManager().isActive(this);
  }

}
```

Schritt 2:

Konfigurationsklasse

Konfiguration

```
public class MyTogglzConfig implements TogglzConfig {

  public Class<? extends Feature> getFeatureClass() {
    return MyFeatures.class;
  }

  public StateRepository getStateRepository() {
    return new FileBasedStateRepository(
                new File("/somewhere/features.properties"));
  }

  public UserProvider getUserProvider() {
    return new ServletUserProvider();
  }

}
```

Spring

```
@Component
public class MyTogglzConfig implements TogglzConfig {

  @Autowired
  private DataSource dataSource;

  public Class<? extends Feature> getFeatureClass() {
    return MyFeatures.class;
  }

  public StateRepository getStateRepository() {
    return new JDBCStateRepository(dataSource);
  }

  public UserProvider getUserProvider() {
    return new SpringSecurityUserProvider("ADMIN");
  }

}
```

Verwendung

```
public void someBusinessMethod() {

  // some code

  if( MyFeatures.FEATURE_ONE.isActive() ) {

    // new code for FEATURE_ONE

  }

  // more code

}
```

JSF Integration


```
<a4j:commandButton execute="@this" value="Reset Sorting"
        action="#{carsSortingBean.reset}" render="table"
        rendered="#{features['FEATURE_ONE']}"
        />
```

Und jetzt?

Wie aktiviere ich nun Features?

- Direkt über die Datei / Datenbank
- Togglz Admin Console

Togglz Admin Console

Was ist die Admin Console?

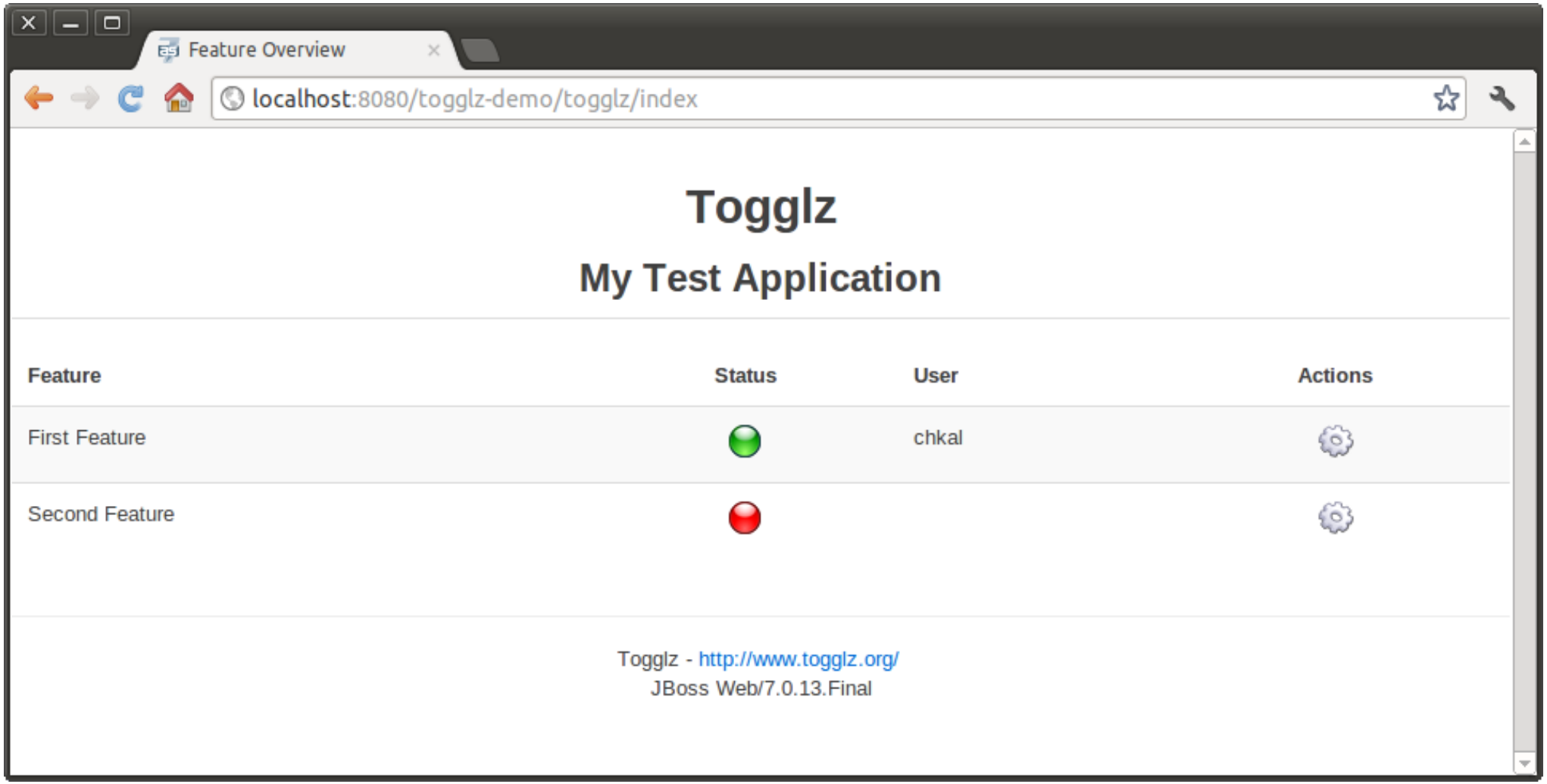
- Integrierte Webanwendung
- Erlaubt es Features zu aktivieren/deaktivieren
- Verwendet Togglz UserProvider

Installation

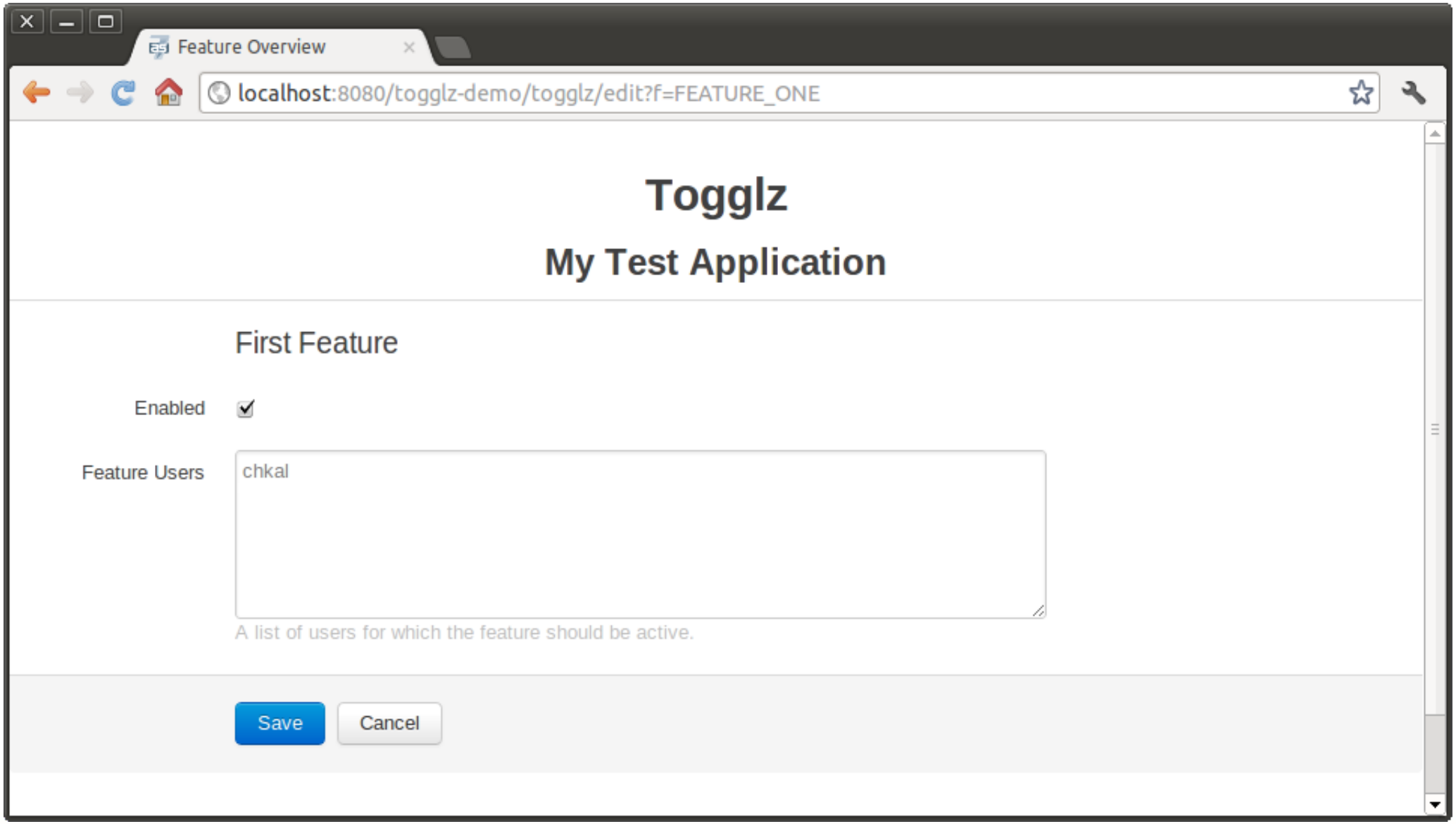
- JAR Archiv zur Applikation hinzufügen
- Fertig!

http://localhost:8080/myapp/togglz

Togglz Admin Console



Togglz Admin Console



Framework Integration

Unterstützte Security APIs

Apache DeltaSpike

Security

```
@Component
public class MyTogglzConfiguration implements TogglzConfig {

  /* ..... */

  @Override
  public UserProvider getUserProvider() {
    return new SpringSecurityUserProvider("ADMIN");
  }

}
```


```
public class MyTogglzConfiguration implements TogglzConfig {

  /* ..... */

  @Override
  public UserProvider getUserProvider() {
    return new SeamSecurityUserProvider();
  }
}

public class SeamSecurityAuthorizer {

  @Secures @FeatureAdmin
  public boolean isFeatureAdmin(Identity identity) {
    // check role
  }
}
```

Erweiterbarkeit

- State Repositories
- User Provider
- Auffinden der Konfigurationklasse
- Komplexe Deployment-Szenarien
- Web-Frameworks?

Zusammenfassung

Togglz ...

- unterstützt den Continuous Delivery Prozess.
- ist einfach zu integrieren / verwenden.
- integriert sich mit bekannten Frameworks.
- ist beliebig erweiterbar.

Fragen?

Vielen Dank für die Aufmerksamkeit!