Aufgaben "Cheesr Onlineshop" (2)

1 Einleitung

Dieses Aufgabenblatt beschreibt den zweiten Aufgabensatz „Cheesr Onlineshop“ für das Wicket. Es basiert auf der Beispielanwendung aus „Wicket in Action“, Manning, 2008.

In diesem Teil des Dokuments werden weitere, auf den Übungen vom vorherigen Teil aufbauende Übungen beschrieben.

2 Übersicht

Im Rahmen der ersten Übung wurde ein Online-Shop entwickelt. Der Online-Shop verfügt über zwei Screens („Index“, „Checkout“), in dem Käse ausgewählt und bestellt werden kann.

Der Musterlösungen mit Beispielcode und alle in den Aufgaben referenzierten Artefakte befinden sich online auf GitHub: https://github.com/anderscore-gmbh/wicket-2020.02

Als Start können Sie das Ergebnis von Aufgabe 8 des vorherigen Blatts verwenden.

3 Vorgehen

Im weiteren Verlauf der Übungsaufgabe wird ein Online-Shop Schritt für Schritt entwickelt. Jeder Schritt behandelt dabei ein Thema. Für jeden Schritt steht eine Beispiellösung in einem Ordner zur Verfügung.

Verwenden Sie den User Guide (ausgedruckt) oder weitere Referenzen (online), um weitere Informationen über die verwendeten Komponenten zu erhalten.

Sollten Sie an einer Stelle nicht weiterkommen oder eine Inspiration benötigen, können Sie gerne in die Beispiellösung schauen oder den entsprechenden Code per Copy&Paste übernehmen.

Hinweise für den zweiten Teil:

- In Abschnitt 4.9 werden Tests im Detail erklärt. Es ist damit gut möglich, folgende Aufgaben Testgetrieben zu entwickeln – dies entspricht durchaus dem Sinn der Aufgaben.
  Mit Ausnahme von Abschnitt 4.9 sind Tests jedoch nicht Bestandteil der Beispiellösung – das Vorgehen bei der Entwicklung ist in erster Linie ihre Entscheidung.
- Die Aufgaben bauen – im Gegensatz zum ersten Teil – kaum aufeinander auf, da bereits ein Grundgerüst vorhanden ist.
  Sie können daher die für Sie interessanten Aufgaben als erstes bearbeiten.

4 Aufgabestellungen

Folgende Aufgaben stehen zur Auswahl

4.8 Best practices / Refactoring: Templating

Aktuell nutzen die Checkout-Page und die Index-Page gleiche Code-Teile für die Anzeige: Auf beiden Seiten wird der Inhalt des Warenkorbs angezeigt. Damit entsteht Redundanz. Nutzen Sie den Templating-Mechanismus, um ein Layout für die Cheesr-Page zu erstellen und Code wiederzuverwenden.

4.8.1 Aufgabenstellung

- Erstellen Sie die Datei CheesrPage.html im gleichen Paket wie die vorhandene Klasse CheesrPage. Übernehmen Sie die gemeinsam genutzten HTML-Bereiche.
  Fügen Sie einen `<wicket:child/>` Tag dort ein, wo die einzelnen Seiten erscheinen sollen.
- Fügen Sie ein `<wicket:extend>`-Element in Index.html und Checkout.html ein.

4.8.2 Hinweise

- Was passiert, wenn die Sub-Page über kein `wicket:extend`-Element verfügt?
- Analog zu Panels werden nur die Bereiche innerhalb des `wicket:extend`-Bereichs in den DOM-Baum eingesetzt.

4.8.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-8-CheesrWebApp

4.9 Testing: Pages und Komponenten

Um die Korrektheit unserer Anwendung auf Knopfdruck überprüfen zu können, sollen im Rahmen dieser Aufgabe diverse Software-Tests geschrieben werden, welche von den speziellen Testing-Mechanismen des Frameworks Gebrauch machen.

Die API-Dokumentation der WicketTester-Klasse gibt eine Übersicht über die zur Verfügung stehenden Methoden.

4.9.1 Aufgabenstellung

Teil 1 (Seitentest – Index und Checkout):

- Legen Sie die zwei Testklassen IndexPageTest und CheckoutPageTest an und binden Sie den PageTester von Wicket ein
- Schreiben Sie einzelne Test-Methoden, welche mindestens folgendes überprüfen:
  - Index: Enthält eine ListView mit der ID „cheeses“
  - Index: Name des ersten dargestellten Käses muss „Gouda“ sein (siehe Backend)
  - Index: ShoppingCart darf nur sichtbar sein, wenn mindestens ein Artikel enthalten ist
  - Checkout: muss ein Formular mit der ID „form“ und ein ShoppingCartPanel mit der ID „cart“ enthalten
  - Checkout: muss ein FeedbackPanel enthalten, welches initial jedoch keinerlei Meldungen anzeigt

Hinweise:

- Erstellen Sie – soweit möglich – auch negativ-Tests.
  - Wie verhält sich die Anwendung, wenn das Backend keine Käsesorten zurückgibt?
  - Was geschieht, wenn die Rückgabeliste ein `nullpointer` ist?
- Beachten Sie API-Dokumentation der Klasse `WicketTester` und Hilfreiche Methoden (vgl. API-Dokumentation):
  - `startPage(Index.class)`
  - `assertRenderedPage(Index.class)`
  - `dumpPage()`
  - `debugComponentTreePage()`
  - `assertComponent("cheeses", ListView.class)`
  - `assertModelValue("cheeses:0:name", "Gouda")`
  - `assertVisible ("checkout")` und `assertInvisible(…)`
  - `clickLink("cheeses:0:add")`
  - `assertNoInfoMessage()` und `assertNoErrorMessage()`

Teil 2 (Komponententest - ShoppingCartPanel):

- Legen Sie die Testklasse ShoppingCartPanelTest an und binden Sie den PageTester von Wicket ein
- Vor der Ausführung einer jeden Testmethode soll stets ein neues ShoppingCartPanel erzeugt und mit zwei exemplarischen Elementen vorbelegt werden
- Schreiben Sie einzelne Test-Methoden, welche mindestens folgendes überprüfen
  - Panel enthält eine ListView mit der ID „cart“
  - Beide vorbelegten Elemente sind durch jeweils zwei Labels (Name und Beschreibung) sowie einen Link (Entfernen) im Panel vertreten
  - Der Gesamtpreis wird in einem Label angezeigt und ist mathematisch korrekt
  - Nach dem Löschen des ersten Warenkorb-Eintrages durch einen Klick auf den zugehörigen Entfernen-Button rückt das zweite Element an die erste Position (sowohl in der GUI als auch im Datenmodell)

Hinweise:

- Hier helfen die folgenden Methoden der Klasse WicketTester (vgl. API-Doc)
  - `startComponentInPage(panel)`
  - `assertLabel("panel:cart:0:name", "Gouda")`

Teil 3 (Markup-Test- ShoppingCartPanel):

- Erweitern Sie nachfolgend die bereits vorhandene Testklasse aus Teil 2
- Bearbeiten Sie die View ShoppingCartPanel.html und ergänzen Sie bei dem tr-Element mit der wicket:id „cart“ die CSS-Klasse „item“
- Schreiben Sie einzelne Test-Methoden, welche den MarkupTester verwenden und mindestens folgendes überprüfen
  - Das Panel enthält für jeden der beiden vorbelegten Warenkorb-Gegenstände ein tr-Element mit der CSS-Klasse „item“
  - Es existiert ein HTML-Element des Typs „tr“ mit der CSS-Klasse „total“ (Gesamtbetrag)

Hinweise:

- Hier helfen die folgenden Methoden des WicketTesters weiter:
  - `tester.assertLabel`
  - `tester.assertComponent`

- Beachten Sie die API-Dokumentation des Tag-Testers um Tags zu testen.

```
1  String responseTxt = wicketTester.getLastResponse().getDocument();
2    List<TagTester> tagTesters =
3      TagTester.createTagsByAttribute(responseTxt, "class", "someClass", false);
4
5    assertEquals(2, tagTesters.size());
```

Teil 4 (Formular-Test- Checkout):

- Legen Sie die Testklasse `CheckoutFormTest` an und binden Sie sowohl `PageTester` als auch `FormTester` von Wicket ein. Nachfolgend testen Sie das Formular auf der Checkout-Page
- Vor der Ausführung der Testmethoden müssen Sie stets die umgebende Page „Checkout“ aufrufen
- Schreiben Sie einzelne Test-Methoden, welche beide Tester verwenden und mindestens folgendes überprüfen
  - Alle Felder des Formulars sind „required“
  - Wenn der Link „cancel“ geklickt wird, so soll die Page Index angezeigt werden
  - Wenn das Formular abgeschickt wird, obwohl ein Feld nicht gefüllt ist, so soll eine Error Message erscheinen
  - Wenn das vollständig ausgefüllte Formular abgeschickt wird, sollen keine Messages zu sehen sein

4.9.2 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-9-CheesrWebApp

4.10 Exceptions behandeln

Trotz umfangreicher Software-Tests besteht grundsätzlich die Möglichkeit, dass zur Laufzeit eines Programms unerwartete Exceptions (insbesondere RuntimeExceptions) auftreten.
Um dem Benutzer keine schwer zu verstehende Fehlermeldung des Application-Servers zuzumuten, ermöglicht Wicket die Definition eigener Error Pages.

4.10.1 Aufgabenstellung

- Legen Sie die Page ErrorPage (HTML + Java) an, die eine kurze, über ein Label angezeigte Fehlermeldung enthält („z.B. Interner Fehler aufgetreten. Bitte kontaktieren Sie den Support“)
  - Erweitern Sie die init-Methode der Klasse `WicketApplication`
    - Anzeigen einer ErrorPage beim Auftreten einer Exception
      Festlegung der zuvor erstellten Page als ErrorPage

4.10.2 Hinweise

Die Anzeige von Exceptions wird in der Klasse Wicket-Application konfiguriert. Folgende Methoden können aus der init-Methode heraus aufgerufen werden:

- `getExceptionSettings().setUnexpectedExceptionDisplay( ExceptionSettings.SHOW_INTERNAL_ERROR_PAGE);`
- `getApplicationSettings().setInternalErrorPage(ErrorPage.class)`

4.10.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-10-CheesrWebApp

4.11 Bookmarkable URLs und Seiten definieren

Einzelne Benutzer wünschen sich eine Detailansicht für die angebotenen Käsesorten und möchten diese auch in Ihrem Browser als Bookmark speichern können. Hierzu konstruieren wir eine Page, die unter einer festen URL erreichbar ist. Der Name der Käsesorte, der in der Detailansicht zu sehen ist, wird durch einen dynamischen Pfadparameter spezifiziert.

4.11.1 Aufgabenstellung

- Legen Sie ein `DetailPanel` an, welches die drei Attribute Name, Beschreibung und Preis eines Käses mit Labels anzeigen kann
- Legen Sie die Page Details (HTML + Java) an
- Der Konstruktor der Page hat ein Argument `PageParameters` und bezieht den Namen der angefragten Käsesorte aus dem Parameter `"name"`.
- Falls der übermittelte Käsename im Pfad zu einer angebotenen Käsesorte aus dem Backend passt, so soll das DetailPanel angezeigt werden.
- Falls hingegen kein passender Käse gefunden wurde, so soll lediglich ein entsprechender Hinweis zu sehen sein.
- Mounten Sie die Details-Page in der init-Methode der Klasse `WicketApplication` unter folgender URL: `/cheese/${name}`.

4.11.2 Hinweise

- Sie können alle verfügbaren Käsesorten von dem BOService beziehen und deren Namen mit dem übermittelten vergleichen.
  In der Praxis würden Sie nicht die vollständige Liste aus dem Backend laden, sondern auf gesonderte finder-Methoden zurückgreifen
- Verschlüsselungsmöglichkeiten und Mount-Optionen für URLs und Ressources werden im Wicket User Guide erklärt (22.4 URLs encryption in detail).
  Optional: Verschlüsseln Sie ihre URLs mit einer einfachen md5-des-Strategie – die Konfiguration erfolgt erneut in der Wicket-Application-Klasse. Verifizieren Sie, dass die URLs obfuskiert sind.
  Argumentieren Sie: Welchen tatsächlichen Sicherheitsgewinn bietet die Verschlüsselung von URLs?

```
1  @Override
2  public void init(){
3      super.init();
4      setRootRequestMapper(new CryptoMapper(getRootRequestMapper(), this));
5      mountPage("/cheese/${name}", Details.class);
6  }
```

4.11.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-11-CheesrWebApp

4.12 Deployment auf Tomcat

Für weitere Tests soll die Anwendung auf Tomcat deployed werden.

4.12.1 Aufgabenstellung

- Installieren Sie apache-tomcat lokal (Version 8).
- Binden Sie die Installation in Eclipse ein
- Starten Sie die Anwendung auf dem Tomcat-Server in Eclipse. Verifizieren Sie, dass die Anwendung korrekt gestartet wird.
- Erstellen Sie mittels maven ein .war file.
- Starten Sie tomcat von der Konsole aus (beenden Sie ihn vorher in eclipse) und deployen Sie das .war-File im „deployment“-Bereich der Webgui oder kopieren Sie es in den webapps-Ordner auf der Festplatte.

4.12.2 Hinweise

- Alle Server sollten in eclipse beendet werden, da sonst mehrere Server versuchen auf Port `8080/tcp` zu starten.
- Die Konfiguration des modes wird in `src/main/webapp/WEB-INF/web.xml` erläutert.

4.12.3 Beispiellösung

Prinzipbedingt gibt es hier keine Beispiellösung. Die Aufgabe wird bei Bedarf besprochen.

4.13 Größe des PageStores beschränken

In der Pre-Production fällt auf, dass der PageStore in der http-Session abgelegt werden soll, damit im Cluster einfacher synchronisiert werden kann. Er könnte dann aber stark anwachsen, da viele Versionen der Seiten gespeichert werden. Die Größe der Session ist zu reduzieren.

4.13.1 Aufgabenstellung

- Ermitteln Sie: Von welcher Seite (Index, Checkout), werden mehrere Versionen erstellt? Hinterfragen Sie, wie viele Versionen der Seite tatsächlich benötigt werden.
- Verwenden Sie den `HttpSessionDataStore` und beschränken Sie die Anzahl der Seiten.
- Fragt der User eine Seite an, die nicht im PageStore enthalten ist, so erhält er eine PageExpired-Fehlermeldung. Definieren Sie eine eigene Fehlerseite, die dem Benutzer die Situation erklärt.

4.13.2 Hinweise

- Die Verwendung und Konfiguration des `HTTPSessionDataStores` wird detailliert im Wicket-Guide, Kapitel „26.1 Page Storing“ beschrieben. Leider hat die Dokumentation evtl. hier einen Fehler: https://issues.apache.org/jira/browse/WICKET-6202 - korrigiertes Codebeispiel:

```
setPageManagerProvider(new DefaultPageManagerProvider(this){
    @Override
    protected IDataStore newDataStore() {
        return new HttpSessionDataStore(getPageManagerContext(),
            new PageNumberEvictionStrategy(2));
    }
});
```

- Analog zur PageExpiredPage können auch weitere Fehlerseiten definiert werden.
- Recherchieren und argumentieren Sie:
  - Welche anderen Umgebungen erfordern die Nutzung des HTTPSession basierten Stores?
  - Welche Cluster-Strategien vermeiden die Synchronisierung der HTTPSession?
  - Was ist bei der Wicket-Funktion `page.setStatelessHint(boolean value)` zu beachten? Was bewirkt sie? Welche Seiten bieten sich dafür an?
    Optional: Implementieren Sie die Details-Seite stateless (Vgl. Aufgabe 0).

4.13.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-13-CheesrWebApp

4.14 Validierungsfehler anpassen & Behaviours

Die Formular-Validierung soll im Sinne der Corporate -Identity angepasst werden.

4.14.1 Aufgabenstellung

Implementieren Sie:

- Bei Textfeldern mit Validierungsfehler ist die CSS-Klasse `error` gesetzt.
- Im CSS: Textfelder der Klasse `error` werden rot umrandet: Beispiel
  `.error { outline: 1px #F00;}`

4.14.2 Hinweise

- Optional: Fügen Sie Textfields, die ausgefüllt werden müssen, dass HTML-Attribut `required` hinzu.
- Für gemeinsames Verhalten bestimmter Komponenten verwendet wicket sogenannte behaviors. Sie werden via `component.add(.)` hinzugefügt und greifen zu verschiedenen Zeitpunkten in den Lebenszyklus der Komponente ein. Das folgende Beispiel beeinflusst die Konfiguration und den auszugebenden Tag.

```
1  public class RequiredFieldBehaviour extends Behavior {
2      @Override
3      public void onComponentTag(Component component, ComponentTag tag) {
4          super.onComponentTag(component, tag);
5
6          // Test: Instanz von FormComponent? -> Casting
7          // Gibt es feedback-Messages?
8          // "class" Attribut lesen (tag.getAttribut)
9          // und modifziert schreiben
10     }
11     @Override
12     public void onConfigure(Component component) {
13         super.onConfigure(component);
14         // Test: Instanz von FormComponent? -> Casting
15         // Required Flag auf true setzen
16 }
```

4.14.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-14-CheesrWebApp

4.15 Custom Feedback Panel: Fehleranzeige beeinflussen

Fehler bei der Validierung sollen dem Benutzer im Sinne der Corporate-Identity angezeigt werden. Hierzu soll für das Rendering des Feedback-Panels die folgende HTML-Struktur verwendet werden.

```
1  <div class="panel panel-danger">
2    <div class="panel-body">
3      Bei der Bestellung sind Fehler aufgetreten
4    </div>
5    <ul>
6      <li>Validierungsfehler</li>
7      <li>Ggf. weitere Validierungsfehler</li>
8      <li>...</li>
9    </ul
10   <div class="panel-footer">Die Bestellung konnte nicht bearbeitet werden.</div>
11 </div>
```

4.15.1 Aufgabenstellung

- Implementieren Sie eine eigenes Feedbackpanel (`CooperateFeedbackPanel extends Feedbackpanel`) mit eigenem HTML-Template.
  Übernehmen Sie dazu den Inhalt der Datei `FeedbackPanel.html` aus den Framework-Sources und passen Sie das Panel an.

4.15.2 Hinweise

- Das Panel kann natürlich auch programmatisch über ein behavior angepasst werden – da die Struktur jedoch recht komplex ist, bietet sich ein eigenes HTML-Template an.
  Eine programmatische Anpassung würde zu HTML-Code innerhalb des Java-Codes führen.
- API und Layout des Feedback-Panels ändern sich ggf. bei künftigen Wicket-Versionen. Um mehr Stabilität zu erreichen, können Sie explizit eine Adapter-Schicht einführen oder ggf. weitere Teile des Frameworks-Codes (z.B. die Message-Darstellung) übernehmen.

4.15.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-15-CheesrWebApp

4.16 Login zur Bestellung – Wicket Autorisation

Nur noch Kunden mit Kundenkonto sollen Käse bestellen dürfen.

4.16.1 Aufgabenstellung

- Fügen Sie das Artefakt `wicket-auth-roles` analog zu wicket (group-id, version) in die `pom.xml` ein.
- Ändern Sie die `CheesrSession`:
  - Sie erbt nun von `AuthenticatedWebSession`.
  - Die Methode `authenticate(String arg0, String arg1)` wird überschrieben. Sie gibt `true` zurück, soweit Benutzer und Kennwort `gfu` sind.
  - Rollen müssen nicht ermittelt werden – die Methode `getRoles()` gibt immer `null` zurück.
- Erstellen Sie ein Interface `AuthenticatedWebPage` um Pages zu taggen, die nur nach einem Login sichtbar sein sollen – taggen Sie `Checkout`, damit hier ein login verlangt wird.
- Erstellen Sie eine Login-Page mit einem Formular für Benutzername / Kennwort.
  - Bei einem Submit des Formulars, wird der Benutzer via `CheesrSession.authenticate(…)` angemeldet.
    - Bei Erfolg, wird der Benutzer auf die letzte besuchte Seite geleitet.
    - Bei Misserfolg wird ein Fehler angezeigt.
- Konfigurieren Sie eine Authorization Strategie in der `init()`-Methode der Wicket-Application und leiten Sie nicht-autorisierte Zugriffe um. Beispiel:

```
1  getSecuritySettings().setAuthorizationStrategy(new IAuthorizationStrategy.AllowAllAuthorizationStrategy() {
2      @Override
3      public <T extends IRequestableComponent> boolean isInstantiationAuthorized(Class<T> componentClass) {
4          if (AuthenticatedWebPage.class.isAssignableFrom(componentClass)) {
5              if (((CheesrSession) Session.get()).isSignedIn()) {
6                  return true;
7              }
8              throw new RestartResponseAtInterceptPageException(SignInPage.class);
9          }
10         return true;
11     }
12 });
```

4.16.2 Hinweise

- Im onSubmit-Handler des Login-Formulars sind folgende Methoden hilfreich:
  - `continueToOriginalDestination()` – die Seite auf der die `Intercept-Exception` auftrat wird erneut aufgerufen
  - `error(String errmsg);` fügt einen Fehler (Freitext) dem Handler hinzu. Ein Hinweis auf einen fehlgeschlagenen Login kann damit angezeigt werden.
- Diese Übungsaufgabe orientiert sich am Codebeispiel authentication1 (http://examples8x.wicket.apache.org/authentication1). Die Beispiele sind Bestandteil der Framework-Dokumentation und zu viele Seiten verfügbar.
- Weitere Hinweise erhalten Sie im Kapitel 22 „Security with Wicket“ im Wicket Guide

4.16.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-16-CheesrWebApp

4.17 Spring Integration

In dieser Übungsaufgabe wird die Integration von Spring in Apache Wicket demonstriert. Typischerweise werden bei Wicket Objekte mit `new` instanziert und serialisiert. Damit dependency-Injection im Framework verwendet werden kann, sind entsprechende Routinen explizit integriert.

In diesem Beispiel wird der Middleware-Client (BOServices) an die passende Stelle injiziert.

4.17.1 Aufgabenstellung

- Binden Sie `wicket-spring` und `spring-context` als dependencies in das Wicket-Projekt ein.
- Erstellen Sie ein Feld BOServices in der Checkoutpage – inkl. getter, setter – und markieren Sie es mit `@Inject`
- Erstellen Sie eine Spring-Kontext-Konfiguration in der Wicket Application
- Verifizieren Sie, dass das Bean korrekt instanziiert werden kann.

4.17.2 Hinweise

- Wicket injiziert die Dependency im normalen Komponenten-Lebenszyklus und nicht auf „magische Weise“ ☺.
- In Tests benötigen Sie ein anderes Vorgehen. Weitere Hinweise siehe Wicket-Guide Kapitel 24 „Test Driven Development with Wicket and Spring“

4.17.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-17-CheesrWebApp

4.18 Internationalisierung durch Locales

Käse wird in Deutschland und Amerika verkauft. Erstellen Sie eine Deutsche und eine Amerikanische Version des Shops

4.18.1 Aufgabenstellung

- Die Auswahl des Lands soll über ein Drop-Down-Feld in der Menüleiste möglich sein.
- Die Preise werden je nach Land in € oder $ angezeigt. Umrechnungskurs: 1:1
- Der Deutsche Titel des Shops ist „cheesr – alles Käse“, der des Amerikanischen „cheesr – cheese online.
- Im A/B Testing wurde festgestellt, dass deutsche Kunden – im Gegensatz zu amerikanischen Kunden – den Bestellvorgang mit einer höheren Wahrscheinlichkeit abschließen, wenn:
  - Die Adresseingabe rechts von der Warenkorb Übersicht erfolgt
  - Die Postleitzahl unterhalb der Stadt eingegeben wird

4.18.2 Hinweise

- Wicket bietet viele Möglichkeiten zur Internationalisierung via Locales. Machen Sie sich mit dem entsprechenden Kapitel im Wicket Guide vertraut.
- Häufig ist es einfacher wartbar, einzelne Strings anstelle ganzer Layouts zu übersetzen. Die Anordnung der Felder wird aber im Layout definiert. Dennoch muss für die deutsche Variante ein lokalisiertes Layout der Checkout-Page erstellt werden, damit die Felder anders angeordnet sind.
- Zur Lokalisierung der Preise können Models verwendet werden. Hier liefert die getObject()-Methode formatierte Daten.
- `java.text.NumberFormat` liefert mit `getCurrencyInstance(Locale local)` eine zur Locale passende Währungsdarstellung.

4.18.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-18-CheesrWebApp

4.19 Konkurrierende Zugriffe: Optimistic Locking & LoadableDetachableModels

Der Shop benötigt eine Administrations-Funktion, mit dem die Daten der Sorten geändert werden können.

4.19.1 Aufgabenstellung

- Erstellen Sie ein Formular, mit dem Name, Beschreibung und Preis geändert werden können.
- Fügen Sie der Listenansicht einen Edit-Link hinzu, mit dem auf das Formular navigiert werden kann
- Nutzen Sie die update-Methode `updateCheese` aus der Klasse `BOService` um den geänderten Käse zu speichern.

4.19.2 Hinweise

- Beim Optimistic Locking führt die Datenbank einen Versionszähler, der mit jedem Schreibvorgang um +1 erhöht wird. Vor dem Speichern wird verglichen, ob der Zähler noch den ursprünglichen Stand hat, oder bereits durch einen anderen Benutzer erhöht wurde. Falls der Zähler geändert wurde, dann schlägt der Speichervorgang fehl.
- Das LoadableDetachableModel kann nicht ohne Anpassung mit OptimisticLocking verwendet werden. Die load()-Methode wird vor dem Speichern ausgeführt und lädt den jeweils aktuellen Satz aus der Datenbank. Damit wird auch der Zählerstand überschrieben.
- Der `BOService` ist recht einfach gehalten und erwartet, dass der ursprüngliche Zähler beim Update übergeben wird. Anderen Frameworks (z.B. Hibernate) erwarten ein anderes Vorgehen.

4.19.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-19-CheesrWebApp

4.20 Dynamisches Layout & nicht-strukturierte Dokumente

Jeder Käsesorte können dynamische Attribute zugewiesen werden (`Cheese::additionalAttributes`), die sich von Käse zu Käse unterscheiden können. Zeigen Sie die Attribute und ihre Werte an.

4.20.1 Aufgabenstellung

- Erweitern Sie die Sorten-Liste auf der Index-Seite: Zeigen Sie zum Käse alle verfügbaren Attribute an.
- Die Sortierreihenfolge kann über die Datei `config.yaml` konfiguriert werden. Ist für ein Attribut keine Sortierreihenfolge hinterlegt, so erscheint es am Ende (Tie-Break: alphabetische Sortierung). Beispiel:

  ```
  cheese_attribute_order:
    - Discount
    - Delivery Date
  ```

  An den ersten Stellen werden die Attribute Discount und Delivery Date angezeigt, dahinter weitere Attribute (z.B. Fat Content).

4.20.2 Hinweise

- Die Beispiellösung enthält bereits einen YAML-Parser und Comparator ohne Wicket-Bindings. Er kann übernommen werden.
- Versuchen Sie, auf Factories oder Factory-Methoden zu verzichten (vgl. Wicket Best Practices)

4.20.3 Beispiellösung

https://github.com/anderscore-gmbh/wicket-2020.02/tree/master/Aufgabe-20-CheesrWebApp

4.21 Tabelle mit CSV Export als Resource

Die Käsesorten sollen als Tabelle dargestellt werden, damit Benutzer einfach nach Spalten sortieren können. Die Tabelle soll über eine Export-Funktion verfügen, die die gleichen Spalten enthält. Eine Suche muss nicht implementiert werden.

4.21.1 Aufgabenstellung

- Ersetzen Sie die ListView auf der Index-Seite durch eine DataTable. Die Tabelle verfügt über 4 Spalten:
    1. Name der Käsesorte
    2. Beschreibung
    3. Preis
    4. „Add To Cart“-Link
- Platzieren Sie einen Link „Export CSV“ oberhalb der Tabelle. Der Export soll die kompletten Daten der Tabelle enthalten (d.h. Pagination nicht berücksichtigen).
- Der Preis soll im Export und in der Ansicht in der Locale des Servers angezeigt werden (`NumberFormat.getCurrencyInstance().format(..)`)

4.21.2 Hinweise

- Die Klasse `DefaultDataTable` ist im Bereich wicket-extras enthalten – einer zusätzlichen Maven Depedency.
- Ein kurzes Beispiel zur Verwendung der DataTable finden Sie unter: https://cwiki.apache.org/confluence/display/WICKET/Simple+Sortable+DataTable+Example – der Code bezieht sich jedoch auf eine alte Wicket Version.
- Verwenden Sie die Definition der Spalten gleichzeitig für Tabelle und Export: Wicket verfügt über einen CSV-Exporter – die Basisklassen und Interfaces sind z.T. identisch. Bspw. kann mit den Klassen `PropertyColumn` und `LambdaColumn` gleichzeitig eine Spalte für den `CSVExporter` und für die DataTable definiert werden.
    - https://ci.apache.org/projects/wicket/apidocs/8.x/org/apache/wicket/extensions/markup/html/repeater/data/table/export/CSVDataExporter.html
    - https://ci.apache.org/projects/wicket/apidocs/8.x/org/apache/wicket/extensions/markup/html/repeater/data/table/LambdaColumn.html
    - https://ci.apache.org/projects/wicket/apidocs/8.x/org/apache/wicket/extensions/markup/html/repeater/data/table/PropertyColumn.html