Gr. 1, E. Pitzer
Gr. 2, F. Gruber-Leitner

Name _____ Aufwand in h _____

Punkte _____ Kurzzeichen Tutor / Übungsleiter _____ / _____

Wie angekündigt wollen wir in den restlichen Übungen – also von der vorliegenden 6. bis zur 8. Übung – anhand *einer* Aufgabenstellung verschiedene Möglichkeiten der Java-Klassenbibliothek ausloten. Damit Sie die weiteren Übungen meistern können, sollten Sie von Anfang an mitmachen.

CaaS – Campina as a Service: Online-Menübestellung

Kurzbeschreibung

Unsere Haus-und-Hof-Mensa möchte für die Studierenden ein besonderes Service anbieten: Um die Wartezeit auf das Mittagsmenü zu minimieren, kann bereits am Vormittag das Wunschmenü online reserviert werden und kommt dann pünktlich heiß auf den Tisch.

Dafür wird die Erstellung einer Software in Auftrag gegeben, die im Wesentlichen folgende Anforderungen erfüllen soll:

Menüverwaltung

- Wartung der aktuellen Speisekarte
- Benutzerverwaltung (hinzufügen, aktualisieren, sperren von Benutzern)
- Übersicht über getätigte Bestellungen

Online-Bestellung

- Login mit Benutzername und Passwort
- Bestellung eines Menüs für den aktuellen Tag

Menüverwaltung

Die Menüverwaltung steht dem Restaurantbetreiber zur Verfügung und ermöglicht ihm, seine aktuelle Speisekarte zu warten. Die Speisekarte besteht aus verschiedenen Bereichen („Vegetarische Gerichte", „Aus der Pfanne und vom Grill", „Fischspezialitäten" etc.). Für die Online-Bestellung werden nur die Hauptspeisen eingetragen, alle anderen Speisen (Desserts, etc.) können ohnehin rasch serviert werden und brauchen nicht berücksichtigt zu werden. Jede Speise besteht aus der Beschreibung („Wiener Schnitzel mit Petersilkartoffeln") und einem Preis. Um auch vorübergehende Angebote abbilden zu können, sollen Speisen optional mit einem Beginn- und Endedatum versehen werden können.

Möchten Studierende das Online-Service nutzen, müssen sie sich beim Restaurantbetreiber persönlich melden. Dieser kann dann in der Menüverwaltung neue Benutzer anlegen (Vorname, Nachname, Benutzername, Passwort). Wenn ein Benutzer mehrmals ein Menü bestellt, ohne es abzuholen, so möchte der Betreiber den Benutzer auch wieder sperren können.

In einer Übersichtsanzeige sieht das Restaurant alle Bestellungen des aktuellen Tages (gereiht nach der Uhrzeit, zu der die Benutzer das Menü serviert bekommen wollen) und kann so punktgenau kochen.

Online-Bestellungsplattform

Nach der Anmeldung mit den erhaltenen Zugangsdaten erscheint die aktuelle Speisekarte. Der Benutzer kann aus der Speisekarte eine Hauptspeise auswählen und muss eine Uhrzeit angeben, zu der die Speise fertig sein soll. Optional kann er auch Sonderwünsche bekannt geben – einen „Gruß *an* die Küche" quasi.

Die Online-Plattform ist allerdings jeden Tag nur bis 11:00 geöffnet, anschließend werden keine Bestellungen mehr angenommen.

Ausbaustufe 1: CaaS-FX   (24 Punkte)

Der Restaurantbetreiber möchte vor der endgültigen Auftragserteilung der Online-Plattform zunächst einen Prototyp für die Menüverwaltung sehen.

Entwickeln Sie daher mit Hilfe von JavaFx für die Menüverwaltung einen funktionstüchtigen, ausbaufähigen Benutzeroberflächen-Prototyp. Versuchen Sie eine möglichst intuitiv zu verwendende Benutzeroberfläche zu entwerfen. Setzen Sie dafür Ihr in UEN erworbenes Wissen ein. Trennen Sie den Entwurfs- vom Implementierungsprozess, indem Sie Ihre Benutzeroberfläche zunächst mit Mockups (Grobentwurf der Benutzeroberfläche) modellieren. Fügen Sie die Mockups zu Ihrer Systemdokumentation hinzu.

Trennen Sie den Code zur Realisierung der grafischen Benutzeroberfläche vom Code zur Repräsentation der Daten Ihrer Anwendung (Benutzer, Menüs, Bestellungen etc.). Durch diese Maßnahme wird Ihre Anwendung einfach erweiterbar, was Ihnen in der nächsten Ausbaustufe zugutekommen sollte. Die Verwendung von FXML und Werkzeugen zum Design der grafischen Benutzeroberfläche (JavaFX Scene Builder) ist nicht erlaubt.

Ihr Prototyp soll es ermöglichen, alle Fenster und Dialoge der Benutzeroberfläche zu öffnen (und diese auch wieder zu schließen). Die Anwendung muss die Eingaben aber noch nicht über die Programmlaufzeit hinaus speichern können. Als Ersatz dafür können Sie mit hart codierten Daten arbeiten. Der Benutzeroberflächen-Prototyp muss folgende Funktionen abdecken:

- Benutzer hinzufügen/sperren, Benutzerdaten aktualisieren
- Speisekarte: Bereiche erstellen/löschen
- Speisekarte: Hauptspeisen hinzufügen/löschen
- Anzeige der heutigen Bestellungen

Beachten Sie, dass die Implementierung der anderen Systemkomponenten erst in weiteren Ausbaustufen gefordert ist. Die genauen technischen Anforderungen an diese Komponenten werden im weiteren Verlauf der Übung bekannt gegeben.

1 Campina as a Service

1.1 Lösungsidee

Folgend ist die Dokumentation für die Aufgabenstellung Campina as a Service angeführt.
Bei dieser Aufgabenstellung soll in Bezug auf die Softwarearchitektur wie folgt vorgegangen werden.

1. Das Datenmodell soll über POJOs abgebildet werden, die nicht direkt verwendet werden.

2. Der Zugriff auf die Datenkomponenten einer Entität innerhalb der Views soll über eigene Models abgebildet werden.

3. Die einzelnen View Teile sollen über eigenen Klassen abgebildet werden, die die View aufbauen un die View Logic beinhalten.

4. Eigenen Kontroller Implementierungen reagieren auf die ActionEvents der Buttons, die Operationen auf den Entitäten anwenden.

5. In der nächsten Stufe sollen die Datenzugriffe über DAOs abgebildet werden, die in den Kontroller Implementierungen verwendet werden.

Folgende Abbildung zeigt den prinzipiellen Aufbau aller Formulare.
Da die Entitäten nur wenige Skalare Datenkomponenten enthalten, abgesehen von den 1-n, ... Beziehungen, soll hier eine einfache Form angewendet werden.
Diese Views können in Zukunft auch noch abgeändert und die List Daten anders dargestellt werden.

Abbildung 1: Formularaufbau

Um den folgenden Aufwand der Formulare zu verringern und die Konsistenz der Fromulare zu erhalten, soll eine Komponente entworfen werden, die in der Lage ist deklarativeKlassen oder Methoden Annotationen bzw. deren enthaltenen Informationen zu verarbeiten und ein Formular zu generieren.

Um zu Vermeiden das wahrlos Referenzen durch die Klassen gezogen werden, soll eine Kontext implementiert werden, der in der Lage ist die Nötigen Informationen und Referenzen zu verwalten, damit sie in den Implementierungen verwendet werden können.

Die Implementierungen werden in Javadoc und in der folgenden Dokumentation beschrieben.

1.2 Source-Code(Implementation View API)

Folgend ist der Source der View API Implementierungen der Übung angeführt.

1.2.1 AbstractViewModel.java

Folgende abstrakte Klasse stellt die root Klasse für alle View Models dar, die eine Entität verwalten.

Listing 1: AbstractViewModel.java

```java
package at.fh.ooe.swe4.fx.campina.view.api;

import java.io.Serializable;
import java.util.Objects;

import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;

/**
 * This model is the base for all entity amanging models.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 * @param <I>
 *            the type of the entity id used in this model
 * @param <T>
 *            the entity type the current instance is managing
 */
public abstract class AbstractViewModel<I extends Serializable, T extends AbstractEntity<I>> {

  private I  id;
  private T  entity;

  /**
   * Initially calls reset()
   *
   * @see AbstractViewModel#reset()
   */
  public AbstractViewModel() {
    super();
    reset();
  }

  /**
   * Initializes this instance with the given entity
   *
   * @param entity
   *            the entity this models backs
   * @throws NullPointerException
   *             if the given entity is null
   */
  public AbstractViewModel(final T entity) {
    prepare(entity);
  }

  /**
   * Resets this model by setting a new entity.<br>
   * This method shall call prepare right away so that this model is correctly
   * initialized.
   */
  public abstract void reset();

  /**
```

```
53       * Prepares this model with the given entity
54       *
55       * @param entity
56       *             the entity this model backs.
57       */
58      public void prepare(T entity) {
59        Objects.requireNonNull(entity);
60
61        setEntity(entity);
62        setId(entity.getId());
63      }
64
65      /**
66       * The model id retrieved from backed entity
67       *
68       * @return the model id
69       */
70      public I getId() {
71        return id;
72      }
73
74      /**
75       * Sets the model id, shall be equal to backed entity id.
76       */
77      public void setId(I id) {
78        this.id = id;
79      }
80
81      /**
82       * Gets the backed entity
83       *
84       * @return the backed entity
85       */
86      public T getEntity() {
87        return entity;
88      }
89
90      /**
91       * Sets the backed entity
92       *
93       * @param entity
94       *             the entity this model backs
95       */
96      public void setEntity(T entity) {
97        this.entity = entity;
98      }
99
100     @Override
101     public int hashCode() {
102       final int prime = 31;
103       int result = 1;
104       result = prime * result + ((id == null) ? 0 : id.hashCode());
105       return result;
106     }
107
108     @Override
109     public boolean equals(Object obj) {
110       if (this == obj) {
111         return true;
112       }
113       if (obj == null) {
114         return false;
115       }
```

```
116      if (getClass() != obj.getClass()) {
117        return false;
118      }
119      AbstractViewModel<I, T> other = (AbstractViewModel<I, T>) obj;
120      if (id == null) {
121        if (other.id != null) {
122          return false;
123        }
124      } else if (!id.equals(other.id)) {
125        return false;
126      }
127      return true;
128    }
129
130 }
```

1.2.2 IdHolder.java

Folgendes Interface markiert eine Klasse oder ein anderes Interface als IdHolder.

Listing 2: IdHolder.java

```java
1  package at.fh.ooe.swe4.fx.campina.view.api;
2
3  /**
4   * Marks an id holder.
5   *
6   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
7   * @date Jun 3, 2015
8   * @param <T>
9   *            the type of the hold id
10  */
11 public interface IdHolder<T> {
12
13    /**
14     * Teh id of this id holder
15     *
16     * @return the id
17     */
18    public T getId();
19
20 }
```

1.2.3 ViewHandler.java

Folgendes Interface spezifiziert einen View Handler, der einen Teilbaum einer Scene verwaltet.

Listing 3: ViewHandler.java

```java
package at.fh.ooe.swe4.fx.campina.view.api;

import javafx.scene.Scene;

/**
 * Specifies an handler for a view or view part.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 3, 2015
 * @param <N>
 *            the type of the component to create and manage
 */
public interface ViewHandler<N> extends IdHolder<String> {

  /**
   * Creates the scene part.
   *
   * @return the root node of this scene part.
   */
  public N createNode();

  /**
   * Initializes this part right after the {@link Scene} holds this part.
   */
  public void initHandler();
}
```

1.2.4 EventHandlerFactory.java

Folgendes Interface spezifiziert eine Event Hadnler Factory die Events registriert und die registrierten Events in Form einer Map liefert.

Listing 4: EventHandlerFactory.java

```java
package at.fh.ooe.swe4.fx.campina.view.api;

import java.io.Serializable;
import java.util.List;
import java.util.Map;

import javafx.event.Event;
import javafx.event.EventHandler;
import javafx.event.EventType;

/**
 * Specifies an EventHandlerFactory which provides a mpa of event handlers
 * mapped to their event Type.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 * @param <T>
 *            the key type
 */
public interface EventHandlerFactory<K> extends Serializable {

    /**
     * registers the events for an given id which should point to an
     * EventTarget.
     *
     * @param id
     *            the event target id
     * @return the map containing the registered events
     */
    public <T extends Event> Map<EventType, List<EventHandler>> registerEventHandler(final K id);
}
```

1.3 Source-Code(Implementation View Annotation)

Folgend ist der Source der Annotation der Übung angeführt.

1.3.1 FormField.java

Folgende Annotation markiert eine Methode als ein Form Field.
Diese Annotation wird dazu verwendet und die Form zu erstellen und zu verwalten.

Listing 5: FormField.java

```java
package at.fh.ooe.swe4.fx.campina.view.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;

/**
 * This annotation marks an method as an form field and connects the form field
 * to the model method.<br>
 * This annotation must be placed on valid java bean getter method.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(value = {
        ElementType.METHOD
})
public @interface FormField {

  /**
   * @return the form field id
   */
  String id();

  /**
   * @return the backed form field type
   */
  FormFieldType type();

  /**
   * @return the form field label
   */
  String label();

  /**
   * @return Message displayed if required but not set. Default ""
   */
  String requiredMessage() default "";

  /**
   * @return Flag which marks an form field as required. Default false
   */
  boolean required() default false;

  /**
   * @return the ordinal of this form field in the form
   */
```

```
52    int ordinal() default 1;
53
54    /**
55     * @return additional value class. Must be used when form field of type
56     *         {@link FormFieldType#SELECT}.
57     */
58    Class<?> valueClass() default Object.class;
59
60    /**
61     * An event handler factory which can be used to provide event listener for
62     * the generate form field
63     */
64    EventHandlerFactories eventHandlerFactories() default @EventHandlerFactories();
65  }
```

1.3.2 SelectFormField.java

Folgende Annotation markiert eine Methode als einen Data Provider für ein Formularfeld vom Type Select.

Listing 6: SelectFormField.java

```java
package at.fh.ooe.swe4.fx.campina.view.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;
import javafx.util.StringConverter;

/**
 * This annotation marks an valid java bean getter method as an data provider
 * for an select field.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface SelectFormField {

  /**
   * @return property name of the form field with type
   *         {@link FormFieldType#SELECT}.<br>
   *         E.g.: target="day" -> getDays
   */
  String target();

  /**
   * @return An converter used for the backed form field of type
   *         {@link FormFieldType#SELECT}.
   */
  Class<? extends StringConverter> converter() default StringConverter.class;
}
```

1.3.3 EventHandlerFactories.java

Folgende Annotation kann in der Formfield Annotation verwendet werden um EventHandlerFactories für diese Formularfeld zu definieren.

Listing 7: EventHandlerFactories.java

```java
package at.fh.ooe.swe4.fx.campina.view.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import at.fh.ooe.swe4.fx.campina.view.api.EventHandlerFactory;

/**
 * This annotation allows to define event listener factories for form field
 * annotated methods.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(value = ElementType.ANNOTATION_TYPE)
public @interface EventHandlerFactories {

  /**
   * @return the array of event handler factories.
   */
  Class<? extends EventHandlerFactory>[] value() default {};
}
```

1.4 Source-Code(Implementation View Component Buidler)

Folgend ist der Source der Komponenten Builder von JavaFX Komponenten der Übung angeführt. Dieser Ansatz ist in Zukunft in Frage zu stellen, ob sich lohnt das erstellen der Komponenten auszulagern.

1.4.1 AbstractFxComponentBuilder.java

Folgende abstrakte Klasse stellt den root aller Builder Implementierungen dar und zwing die abgeleiteten Klassen für einen konkreten Typ der Builder zu sein.

Listing 8: AbstractFxComponentBuilder.java

```java
package at.fh.ooe.swe4.fx.campina.component.builder.api;

import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Objects;

import javafx.event.EventHandler;
import javafx.event.EventType;

/**
 * This is the base builder for all component buidlers.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 * @param <T>
 *            the Component this builder is for
 * @param <B>
 *            the actual buidler type
 */
@SuppressWarnings({
        "unchecked", "rawtypes" })
public abstract class AbstractFxComponentBuilder<T, B extends AbstractFxComponentBuilder<T, B>> {

  public Map<EventType, List<EventHandler>>  events;

  /**
   *
   */
  public AbstractFxComponentBuilder() {
  }

  /**
   * Registers the event handlers
   *
   * @param handlers
   *            the handlers to register
   * @return the current instance
   */
  public B registerEventHandlers(
      final Map<EventType, List<EventHandler>> handlers) {
    checkIfStarted();
    Objects.requireNonNull(handlers, "Cannot add event handlers because provided map is null");

    for (Entry<EventType, List<EventHandler>> entry : handlers.entrySet()) {
      events.put(entry.getKey(), entry.getValue());
    }

```

```
49        return (B) this;
50    }
51
52    /**
53     * Ensures that the handler has been porperly started.
54     */
55    protected abstract void checkIfStarted();
56
57 }
```

1.4.2 DuplicateKeyException.java

Folgende Klasse stellt die Exceptiond ar um anzuzeigen ob sich doppelte Ids in der Scene befinden. Wird aber nur im Buidler verwendet, da es mir schwer viel alle Ids in allen Handlern zu überprüfen um sicherzustellen dass keine Duplikate bezüglich den Ids auftreten.

Listing 9: DuplicateKeyException.java

```java
package at.fh.ooe.swe4.fx.campina.component.builder.exception;

public class DuplicateKeyException extends RuntimeException {

    private static final long serialVersionUID = 99858106221010346L;

    public DuplicateKeyException() {
// TODO Auto-generated constructor stub
    }

    public DuplicateKeyException(String message) {
super(message);
// TODO Auto-generated constructor stub
    }

    public DuplicateKeyException(Throwable cause) {
super(cause);
// TODO Auto-generated constructor stub
    }

    public DuplicateKeyException(String message, Throwable cause) {
super(message, cause);
// TODO Auto-generated constructor stub
    }

    public DuplicateKeyException(String message, Throwable cause,
      boolean enableSuppression, boolean writableStackTrace) {
super(message, cause, enableSuppression, writableStackTrace);
// TODO Auto-generated constructor stub
    }

}
```

1.4.3   MenuBarBuilder.java

Folgende Klasse stellt den Builder für MenuBar dar.

Listing 10: MenuBarBuidler.java

```java
package at.fh.ooe.swe4.fx.campina.component.builder.impl;

import java.util.HashSet;
import java.util.Objects;
import java.util.Set;

import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import at.fh.ooe.swe4.fx.campina.component.builder.api.AbstractFxComponentBuilder;

/**
 * This is the buidler for menu bars
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class MenuBarBuilder extends AbstractFxComponentBuilder<MenuBar, MenuBarBuilder> {

  private Set<Menu>  menus;

  /**
   * Starts the buidler
   *
   * @return the current instance
   */
  public MenuBarBuilder start() {
    if (menus != null) {
      throw new IllegalStateException("Builder already initialized");
    }
    menus = new HashSet<>();
    return this;
  }

  /**
   * Ads an menu to this builder
   *
   * @param menu
   *            the menu for the menu bar
   * @return the current instance
   */
  public MenuBarBuilder addMenu(final Menu menu) {
    checkIfStarted();
    Objects.requireNonNull(menu, "Cannto add null menu to menu bar");

    menus.add(menu);
    return this;
  }

  /**
   * Buidls anew menu bar isntance with the ormerly set menus.
   *
   * @return the menu bar instance
   */
  public MenuBar build() {
    checkIfStarted();
    final MenuBar bar = new MenuBar();
    bar.getMenus()
       .addAll(menus);
```

```
59        return bar;
60      }
61
62      /**
63       * Ends this builder. Will need to be restarted if reused.
64       *
65       * @return the current instance
66       */
67      public MenuBarBuilder end() {
68        checkIfStarted();
69        menus = null;
70        return this;
71      }
72
73      @Override
74      protected void checkIfStarted() {
75        if (menus == null) {
76          throw new IllegalStateException("Buuilder not inititalized");
77        }
78      }
79    }
```

1.4.4 MenuBuilder.java

Folgende Klasse stellt den Builder für Menu dar.

Listing 11: MenuBuidler.java

```java
package at.fh.ooe.swe4.fx.campina.component.builder.impl;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map.Entry;
import java.util.Objects;

import javafx.event.EventHandler;
import javafx.event.EventType;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuItem;
import at.fh.ooe.swe4.fx.campina.component.builder.api.AbstractFxComponentBuilder;

/**
 * This is the buidler for {@link Menu}
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class MenuBuilder extends AbstractFxComponentBuilder<Menu, MenuBuilder> {

  public List<MenuItem>  items  = null;

  /**
   * Starts the builder
   *
   * @return the current instance
   */
  public MenuBuilder start() {
    if (items != null) {
      throw new IllegalStateException("Builder already initialized");
    }

    items = new ArrayList<>();
    events = new HashMap<>();
    return this;
  }

  /**
   * Ends the builder. Needs to be restarted i reused.
   *
   * @return the current instance
   */
  public MenuBuilder end() {
    checkIfStarted();
    items = null;
    return this;
  }

  /**
   * Builds the menu instance.
   *
   * @param id
   *            the id of the menu instance
   * @param label
   *            the label of the menu instance
   * @return the menu instance
```

```java
59      */
60   public Menu build(final String id, final String label) {
61     checkIfStarted();
62     Objects.requireNonNull(label, "Cannot create menu with null id");
63     Objects.requireNonNull(label, "Cannot create menu with null label");
64
65     final Menu menu = new Menu(label);
66     menu.setId(id);
67     for (MenuItem item : items) {
68       menu.getItems()
69         .add(item);
70       item.setId(new StringBuilder(item.getParentMenu()
71                       .getId()).append(":")
72                           .append(item.getId())
73                           .toString());
74     }
75
76     for (Entry<EventType, List<EventHandler>> event : events.entrySet()) {
77       for (EventHandler handler : event.getValue()) {
78         menu.addEventHandler(event.getKey(), handler);
79       }
80     }
81     return menu;
82   }
83
84   /**
85    * Adds an item to this builder
86    *
87    * @param item
88    *            the item to be added
89    * @return the current instance
90    */
91   public MenuBuilder addItem(final MenuItem item) {
92     checkIfStarted();
93     Objects.requireNonNull(item, "Cannot add nul item");
94
95     items.add(item);
96     return this;
97   }
98
99   @Override
100  protected void checkIfStarted() {
101    if (items == null) {
102      throw new IllegalStateException("Builder not initialized");
103    }
104  }
105 }
```

1.4.5 MenuItemBuilder.java

Folgende Klasse stellt den Builder für MenuItem dar.

Listing 12: MenuItemBuidler.java

```java
package at.fh.ooe.swe4.fx.campina.component.builder.impl;

import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map.Entry;
import java.util.Objects;
import java.util.Set;

import javafx.event.EventHandler;
import javafx.event.EventType;
import javafx.scene.control.MenuItem;
import at.fh.ooe.swe4.fx.campina.component.builder.api.AbstractFxComponentBuilder;
import at.fh.ooe.swe4.fx.campina.component.builder.exception.DuplicateKeyException;

public class MenuItemBuilder extends AbstractFxComponentBuilder<MenuItem, MenuItemBuilder> {

  private Set<String>  ids;

  public MenuItemBuilder start() {
    ids = new HashSet<String>();
    events = new HashMap<>();
    return this;
  }

  public MenuItemBuilder removeFromContext(final String id) {
    checkIfStarted();
    Objects.requireNonNull(id,
              "Cannot remove item fron context with null id");

    ids.remove(id);
    return this;
  }

  public MenuItem build(final String id, final String label) {
    checkIfStarted();
    Objects.requireNonNull(id, "Cannot create item for null id");
    Objects.requireNonNull(label, "Cannot create item for null label");

    if (!ids.add(id)) {
      throw new DuplicateKeyException(
          "Builder has already built an item with this id");
    }

    final MenuItem item = new MenuItem(label);
    item.setId(id);
    for (Entry<EventType, List<EventHandler>> event : events.entrySet()) {
      for (EventHandler handler : event.getValue()) {
        item.addEventHandler(event.getKey(), handler);
      }
    }
    return item;
  }

  public MenuItemBuilder end() {
    ids = null;
    events = null;
    return this;
```

```
59      }
60
61      @Override
62      protected void checkIfStarted() {
63        if (ids == null) {
64          throw new IllegalStateException("Builder not inititalized");
65        }
66      }
67    }
```

1.5 Source-Code(Implementation Form Handler)

Folgend ist der Source des Form Handling angeführt.

1.5.1 FormUtils.java

Folgende Klasse stellt Utilities für das Form Handling zur Verfügung.

Listing 13: FormUtils.java

```java
package at.fh.ooe.swe4.fx.campina.view.form;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.text.NumberFormat;
import java.time.Instant;
import java.time.LocalDate;
import java.time.ZoneId;
import java.util.Calendar;
import java.util.Date;
import java.util.Objects;

import javafx.scene.Node;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.DatePicker;
import javafx.scene.control.TextField;
import javafx.util.converter.BigDecimalStringConverter;

/**
 * Utility class for handling form fields.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class FormUtils {

  private FormUtils() {
  }

  /**
   * Enumeration whcih specifies the supported JavaFX components.
   *
   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
   * @date Jun 5, 2015
   */
  public enum FormFieldType {
    INPUT_TEXT(String.class, TextField.class),
    DATE_PICKER(Calendar.class, DatePicker.class),
    DECIMAL(BigDecimal.class, TextField.class),
    SELECT(Object.class, ChoiceBox.class);

    public final Class<?>        valueClass;
    public final Class<? extends Node>  nodeClass;

    /**
     * @param valueClass
     * @param nodeClass
     */
    private FormFieldType(Class<?> valueClass, Class<? extends Node> nodeClass) {
      this.valueClass = valueClass;
      this.nodeClass = nodeClass;
    }
```

```
53    }
54
55    /**
56     * Interface which marks an Validator
57     *
58     * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
59     * @date Jun 5, 2015
60     * @param <T>
61     *            the {@link Node} type this validator is for
62     */
63    public static interface FormFieldValidator<T extends Node> {
64      /**
65       * Answers the question if the current instance is valid.
66       *
67       * @param type
68       * @param inst
69       * @return true if valid, false otherwise
70       */
71      public boolean valid(FormFieldType type, T inst);
72    }
73
74    /**
75     * A required validator for Form Fields.
76     *
77     * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
78     * @date Jun 5, 2015
79     * @param <T>
80     *            the {@link Node} type fo the to validate node
81     */
82    public static class RequiredFormFieldValidator<T extends Node> implements FormFieldValidator<T>
      ↪    {
83
84      @Override
85      public boolean valid(FormFieldType type, T node) {
86        final Object value = getFormFieldValue(type, node);
87        if (value == null) {
88          return Boolean.FALSE;
89        }
90        if (value instanceof String) {
91          return !((String) value).trim()
92                      .isEmpty();
93        }
94        return Boolean.TRUE;
95      }
96    }
97
98    /**
99     * Creates a node regarding the given {@link FormFieldType}
100    *
101    * @param type
102    *            the type of the form field
103    * @return the new corresponding node instance
104    */
105   public static Node create(final FormFieldType type) {
106     Objects.requireNonNull(type);
107
108     switch (type) {
109     case INPUT_TEXT:
110       return new TextField();
111     case DATE_PICKER:
112       return new DatePicker();
113     case DECIMAL:
114       return new TextField();
```

```
115        case SELECT:
116          return new ChoiceBox<>();
117        default:
118          throw new IllegalArgumentException("FormFieldType: '" + type.name() + "' is not managed
           ↪   here");
119      }
120    }
121
122    /**
123     * Gets the node value.
124     *
125     * @param type
126     *            the type of the form field
127     * @param node
128     *            the node to get value from
129     *
130     * @return the node's value
131     * @throws NullPointerException
132     *             if the given node is null
133     * @throws IllegalArgumentException
134     *             if the node is of an unmanaged type
135     */
136    public static Object getFormFieldValue(FormFieldType type, final Node node) {
137      Objects.requireNonNull(node, "Need node to get its value");
138
139      switch (type) {
140      case INPUT_TEXT:
141        return ((TextField) node).getText();
142      case DATE_PICKER:
143        final DatePicker picker = (DatePicker) node;
144        final Calendar cal;
145        LocalDate localDate = picker.getValue();
146        if (localDate != null) {
147          Instant instant = Instant.from(localDate.atStartOfDay(ZoneId.systemDefault()));
148          cal = Calendar.getInstance();
149          cal.setTime(Date.from(instant));
150        } else {
151          cal = null;
152        }
153        return cal;
154      case DECIMAL:
155        final String value = ((TextField) node).getText();
156        if ((value != null) && (!value.trim()
157                       .isEmpty())) {
158          return new BigDecimal(value);
159        }
160        return (BigDecimal) null;
161      case SELECT:
162        final ChoiceBox<?> box = (ChoiceBox<?>) node;
163        return box.getSelectionModel()
164              .getSelectedItem();
165      default:
166        break;
167      }
168      throw new IllegalArgumentException("Nod of type '" + node.getClass()
169                             .getName() + "' unknown");
170    }
171
172    /**
173     * Resets the node value by setting it to null.
174     *
175     * @param type
176     *            the form field type
```

```
177     * @param node
178     *            the node to be reset
179     * @see FormFieldType#setFormValue(Node, Object)
180     */
181    public static void resetFormValue(final FormFieldType type, final Node node) {
182      setFormValue(type, node, null);
183    }
184
185    /**
186     * Sets a value on the given node.
187     *
188     * @param type
189     *            the form field type
190     * @param node
191     *            the node to set value on
192     * @param value
193     *            the value to be set. (Model held type)
194     * @throws NullPointerException
195     *             if the given node is null
196     * @throws IllegalArgumentException
197     *             if the node is of an unmanaged type
198     */
199    public static void setFormValue(final FormFieldType type, final Node node, final Object value) {
200      Objects.requireNonNull(node, "Need node to get its value");
201
202      switch (type) {
203
204      case INPUT_TEXT:
205        ((TextField) node).setText((String) value);
206        return;
207
208      case DATE_PICKER:
209        final LocalDate localDate;
210        if (value != null) {
211          Date date = new Date();
212          Instant instant = date.toInstant();
213          localDate = instant.atZone(ZoneId.systemDefault())
214                  .toLocalDate();
215        } else {
216          localDate = null;
217        }
218        ((DatePicker) node).setValue(localDate);
219        return;
220      case DECIMAL:
221        String formatted = null;
222        if (value != null) {
223          final NumberFormat nf = NumberFormat.getCurrencyInstance();
224          nf.setMaximumFractionDigits(2);
225          nf.setMinimumFractionDigits(1);
226          nf.setRoundingMode(RoundingMode.UNNECESSARY);
227          nf.setMinimumIntegerDigits(1);
228          nf.setMaximumFractionDigits(3);
229          formatted = nf.format(((BigDecimal) value).doubleValue());
230          formatted = new BigDecimalStringConverter().toString((BigDecimal) value);
231        }
232        ((TextField) node).setText(formatted);
233        return;
234      case SELECT:
235        final ChoiceBox<Object> box = (ChoiceBox<Object>) node;
236        box.getSelectionModel()
237          .select(box.getItems()
238                .indexOf(value));
239        return;
```

```
240        default:
241        }
242        throw new IllegalArgumentException("Nod of type '" + node.getClass()
243                                    .getName() + "' unknown");
244    }
245 }
```

1.5.2 FormHandler.java

Dieser Handler erstellt das Formular und verwaltet dieses Formular.

Listing 14: FormHandler.java

```java
package at.fh.ooe.swe4.fx.campina.view.form;

import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;

import javafx.collections.ObservableList;
import javafx.scene.Node;
import javafx.scene.control.ChoiceBox;
import javafx.scene.layout.ColumnConstraints;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Text;
import javafx.util.StringConverter;
import at.fh.ooe.swe4.fx.campina.view.annotation.FormField;
import at.fh.ooe.swe4.fx.campina.view.annotation.SelectFormField;
import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;
import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldValidator;
import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.RequiredFormFieldValidator;

/**
 * This class is a for field handler which handles the form fields related to
 * the model class the handler is for. <br>
 * This implementation uses fluent api.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date May 31, 2015
 * @param <T>
 */
public class FormHandler<T extends AbstractViewModel> {

    /**
     * Model which holds the resolved form field information.
     *
     * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
     * @date May 31, 2015
     */
    private static class FormFieldResolvedModel {

        public final String          id;
        public final String          globalPrefix;
        public final String          methodGetterName;
        public final String          methodSetterName;
        public final FormField       field;
        private final Map<Object, Object>  additionalData  = new HashMap<>();

        /**
         * Used for lightweight instance for searching in set
         *
         * @param globalPrefix
         * @param methodName
         */
```

```java
59     public FormFieldResolvedModel(String globalPrefix, String methodName) {
60         super();
61         this.globalPrefix = globalPrefix;
62         this.methodGetterName = methodName;
63         this.methodSetterName = methodName.replace("get", "set");
64         this.id = "field-" + methodName.substring(3, methodName.length())
65                         .toLowerCase();
66
67         this.field = null;
68     }
69
70     /**
71      * @param globalPrefix
72      * @param methodName
73      * @param field
74      */
75     public FormFieldResolvedModel(String globalPrefix, String methodName, FormField field) {
76         super();
77         Objects.requireNonNull(globalPrefix);
78         Objects.requireNonNull(methodName);
79         Objects.requireNonNull(field);
80
81         if (!methodName.startsWith("get")) {
82             throw new IllegalArgumentException("Method must be a valid java bean getter");
83         }
84         this.globalPrefix = globalPrefix;
85         this.methodGetterName = methodName;
86         this.methodSetterName = methodName.replace("get", "set");
87         this.id = "field-" + methodName.substring(3, methodName.length())
88                         .toLowerCase();
89         ;
90         this.field = field;
91     }
92
93     public String toMessageId() {
94         Objects.requireNonNull(id);
95
96         return globalPrefix + "-message-" + id;
97     }
98
99     public String toLabelId() {
100        return globalPrefix + "-label-" + id;
101    }
102
103    public String toNodeId() {
104        return globalPrefix + "-node-" + id;
105    }
106
107    public <T> void putData(final Object
108            key, T instance) {
109        Objects.requireNonNull(key);
110        Objects.requireNonNull(instance);
111
112        additionalData.put(key, instance);
113    }
114
115    public <T> T getData(final Object key) {
116        Objects.requireNonNull(key);
117
118        return (T) additionalData.get(key);
119    }
120
121    @Override
```

```
122    public int hashCode() {
123      final int prime = 31;
124      int result = 1;
125      result = prime * result + ((id == null) ? 0 : id.hashCode());
126      return result;
127    }
128
129    @Override
130    public boolean equals(Object obj) {
131      if (this == obj)
132        return true;
133      if (obj == null)
134        return false;
135      if (getClass() != obj.getClass())
136        return false;
137      FormFieldResolvedModel other = (FormFieldResolvedModel) obj;
138      if (id == null) {
139        if (other.id != null)
140          return false;
141      } else if (!id.equals(other.id))
142        return false;
143      return true;
144    }
145  }
146
147  private boolean  started  = Boolean.FALSE;
148
149  /**
150   * Empty default constructor
151   */
152  public FormHandler() {
153    super();
154  }
155
156  /**
157   * Initializes this builder.
158   *
159   * @return the current instance
160   * @throws NullPointerException
161   *                 if the given model class is null
162   * @throws IllegalStateException
163   *                 if the builder is already started
164   */
165  public FormHandler<T> init() {
166    if (started) {
167      throw new IllegalStateException("Handler needs to be end before restarted");
168    }
169
170    started = Boolean.TRUE;
171
172    return this;
173  }
174
175  /**
176   * Ands this builder.
177   *
178   * @return the current instance
179   * @throws IllegalStateException
180   *                 if the builder hasn't been started before
181   */
182  public FormHandler<T> end() {
183    checkIfStarted();
184
```

```
185        this.started = Boolean.FALSE;
186        return this;
187    }
188
189    /**
190     * Generates the grid which holds the form and its fields.
191     *
192     * @param ctx
193     *            the backing form context which gets set on each form field
194     *            node (label, field, message)
195     * @return the generated grid
196     * @throws NullPointerException
197     *             if the given id is null
198     * @throws IllegalStateException
199     *             if the builder hasn't been started before
200     */
201    public GridPane generateFormGrid(FormContext<T> ctx) {
202        checkIfStarted();
203        Objects.requireNonNull(ctx, "Form Context must be given");
204
205        // column constrains, should be replaced by css
206        final ColumnConstraints labelColConst = new ColumnConstraints(2);
207        labelColConst.setPrefWidth(150);
208        final ColumnConstraints valueColConst = new ColumnConstraints(2);
209        valueColConst.setPrefWidth(250);
210        final ColumnConstraints messageColConst = new ColumnConstraints(2);
211        valueColConst.setPrefWidth(250);
212
213        // the grid which holds the form
214        final GridPane gridPane = new GridPane();
215        gridPane.setId(ctx.id + "-form");
216
217        gridPane.setHgap(10);
218        gridPane.setVgap(10);
219        gridPane.setUserData(ctx);
220        gridPane.getColumnConstraints()
221            .add(labelColConst);
222        gridPane.getColumnConstraints()
223            .add(valueColConst);
224        gridPane.getColumnConstraints()
225            .add(messageColConst);
226
227        // the form fields defined in the model
228        final List<FormFieldResolvedModel> models = createResolvedModels(ctx);
229
230        // generate form fields
231        for (int i = 0; i < models.size(); i++) {
232            final FormFieldResolvedModel model = models.get(i);
233
234            // form field label
235            final Text labelText = new Text(model.field.label());
236            labelText.setId(model.toLabelId());
237            labelText.setUserData(ctx);
238
239            // form field message
240            final Text messageText = new Text("");
241            messageText.setId(model.toMessageId());
242            messageText.setVisible(Boolean.FALSE);
243            messageText.setUserData(ctx);
244
245            // form field
246            final Node node = FormUtils.create(model.field.type());
247            node.setId(model.toNodeId());
```

```
248        node.setUserData(ctx);
249        // Handing for select type
250        if (model.field.type()
251                .equals(FormFieldType.SELECT)) {
252          final ObservableList<Object> list = model.getData(model.id);
253          final SelectFormField select = model.getData(SelectFormField.class);
254          StringConverter<Object> converter = null;
255          if (!select.converter()
256                  .equals(StringConverter.class)) {
257            try {
258              converter = select.converter()
259                      .newInstance();
260            } catch (Throwable e) {
261              throw new IllegalStateException("Could not isntantiate select string converter '" +
                  ↪  select.converter()
262                                                      .getName() + "'", e);
263            }
264          }
265          final ChoiceBox<Object> box = (ChoiceBox<Object>) node;
266          box.setItems(list);
267          if (converter != null) {
268            box.setConverter(converter);
269          }
270        }
271
272        // TODO: register form field events
273
274        // set form fields on grid
275        gridPane.add(labelText, 0, i);
276        gridPane.add(node, 1, i);
277        gridPane.add(messageText, 2, i);
278      }
279
280      return gridPane;
281    }
282
283    /**
284     * Fills the model with the form field set values.
285     *
286     * @param ctx
287     *          the form context
288     *
289     * @return the current instance
290     * @throws NullPointerException
291     *          if either the scene or model is null
292     * @throws IllegalStateException
293     *          if the model defines form fields which could not be found on
294     *          the given scene.<br>
295     *          If the reflective invocation of the models setter fails<br>
296     *          If the builder hasn't been started yet
297     */
298    public FormHandler<T> fillModel(FormContext<T> ctx) {
299      checkIfStarted();
300      Objects.requireNonNull(ctx, "Need form context to search for form fields");
301
302      final List<FormFieldResolvedModel> models = createResolvedModels(ctx);
303      for (FormFieldResolvedModel fieldModel : models) {
304        final Node node = ctx.scene.lookup("#" + fieldModel.toNodeId());
305        if (node == null) {
306          throw new IllegalStateException("Scene does not contain form field with id");
307        }
308        try {
309          final Method method;
```

```java
310        Class<?> valueClass = fieldModel.field.valueClass();
311        if (valueClass.equals(Object.class)) {
312          valueClass = fieldModel.field.type().valueClass;
313        }
314        method = ctx.model.getClass()
315                    .getMethod(fieldModel.methodSetterName, valueClass);
316        method.invoke(ctx.model, FormUtils.getFormFieldValue(fieldModel.field.type(), node));
317      } catch (Throwable e) {
318        throw new IllegalStateException("Could not set model value", e);
319      }
320    }
321    return this;
322  }
323
324  /**
325   * Fills the form with the model provided set values.
326   *
327   * @param ctx
328   *            the form context
329   *
330   * @return the current instance
331   * @throws NullPointerException
332   *            if either the scene or model is null
333   * @throws IllegalStateException
334   *            if the model defines form fields which could not be found on
335   *            the given scene.<br>
336   *            If the reflective invocation of the models getter fails<br>
337   *            If the builder hasn't been started yet
338   */
339  public FormHandler<T> fillForm(FormContext<T> ctx) {
340    checkIfStarted();
341    Objects.requireNonNull(ctx, "Need context to search for form fields");
342
343    final List<FormFieldResolvedModel> models = createResolvedModels(ctx);
344    for (FormFieldResolvedModel fieldModel : models) {
345      final Node node = ctx.scene.lookup("#" + fieldModel.toNodeId());
346      if (node == null) {
347        throw new IllegalStateException("Scene does not contain form field with id");
348      }
349      try {
350        if (FormFieldType.SELECT.equals(fieldModel.field.type())) {
351          final ChoiceBox<Object> box = (ChoiceBox<Object>) node;
352          box.setItems(fieldModel.getData(fieldModel.id));
353        }
354        final Object value = ctx.model.getClass()
355                      .getMethod(fieldModel.methodGetterName)
356                      .invoke(ctx.model);
357        FormUtils.setFormValue(fieldModel.field.type(), node, value);
358      } catch (Throwable e) {
359        throw new IllegalStateException("Could not set model value", e);
360      }
361    }
362    return this;
363  }
364
365  /**
366   * Resets the form by searching all form elements in the given scene. It
367   * sets all values to null.
368   *
369   * @param ctx
370   *            the form context
371   *
372   * @return the current instance
```

```java
373      * @throws NullPointerException
374      *              if the context is null
375      * @throws IllegalStateException
376      *              if a model has a for field defined but it could not be found
377      *              on the scene
378      */
379     public FormHandler<T> resetForm(FormContext<T> ctx) {
380       checkIfStarted();
381       Objects.requireNonNull(ctx.scene, "Need to scene to search for form fields");
382
383       final List<FormFieldResolvedModel> models = createResolvedModels(ctx);
384       for (FormFieldResolvedModel fieldModel : models) {
385         final Node node = ctx.scene.lookup("#" + fieldModel.toNodeId());
386         if (node == null) {
387           throw new IllegalStateException("Scene does not contain form field with id");
388         }
389         FormUtils.resetFormValue(fieldModel.field.type(), node);
390         final Text messageNode = (Text) ctx.scene.lookup("#" + fieldModel.toMessageId());
391         messageNode.setVisible(Boolean.FALSE);
392       }
393       return this;
394     }
395
396     /**
397      * Validates the form backed by the given {@link FormContext}. <br>
398      * Manipulates the context valid falg during the validation
399      *
400      * @param ctx
401      *              the form context
402      * @return the current instance
403      * @throws NullPointerException
404      *              if the context is null
405      * @throws IllegalStateException
406      *              if the builder hasn't been start yet
407      */
408     public FormHandler<T> validateForm(final FormContext<T> ctx) {
409       checkIfStarted();
410       Objects.requireNonNull(ctx, "Need form context to validate form");
411
412       // expected valid
413       ctx.valid = Boolean.TRUE;
414       // required validator
415       final FormFieldValidator<Node> requiredValidator = new RequiredFormFieldValidator<Node>();
416       // the form field models
417       final List<FormFieldResolvedModel> models = createResolvedModels(ctx);
418       for (FormFieldResolvedModel fieldModel : models) {
419         final Node node = ctx.scene.lookup("#" + fieldModel.toNodeId());
420         if (node == null) {
421           throw new IllegalStateException("Scene does not contain form field with id");
422         }
423         final Text messageNode = (Text) ctx.scene.lookup("#" + fieldModel.toMessageId());
424         // need type validation
425         if (FormFieldType.DECIMAL.equals(fieldModel.field.type())) {
426           try {
427             FormUtils.getFormFieldValue(FormFieldType.DECIMAL, node);
428             messageNode.setVisible(Boolean.FALSE);
429             messageNode.setText("");
430           } catch (NumberFormatException e) {
431             ctx.valid = Boolean.FALSE;
432             messageNode.setVisible(Boolean.TRUE);
433             messageNode.setText("Keine gültige Nummer");
434             continue;
435           }
```

```
436            }
437            // need required validation
438            if (fieldModel.field.required()) {
439              // is invalid
440              if (!requiredValidator.valid(fieldModel.field.type(), node)) {
441                ctx.valid = Boolean.FALSE;
442                messageNode.setVisible(Boolean.TRUE);
443                messageNode.setText(fieldModel.field.requiredMessage());
444              }
445              // reset if valid
446              else {
447                messageNode.setText("");
448              }
449            }
450          }
451          return this;
452        }
453
454        /**
455         * Throws exception if builder is not started when this method gets called.
456         *
457         * @throws IllegalStateException
458         *              if the builder hasn't been started yet
459         */
460        private void checkIfStarted() {
461          if (!started) {
462            throw new IllegalStateException("Builder not started");
463          }
464        }
465
466        /**
467         * Creates a list of resolved field models for the given model class.
468         *
469         * @param ctx
470         *              The context which holds a model to generate the field models
471         *              from
472         * @return the resolved form fields represented by the created model
473         * @throws NullPointerException
474         *              if the context is null
475         */
476        private List<FormFieldResolvedModel> createResolvedModels(FormContext<T> ctx) {
477          Objects.requireNonNull(ctx, "Contextmust not be null");
478
479          final T model = ctx.model;
480          final List<FormFieldResolvedModel> models = new ArrayList<>();
481          final Method[] methods = model.getClass()
482                          .getDeclaredMethods();
483
484          // form field annotations
485          for (Method method : methods) {
486            final FormField field = method.getAnnotation(FormField.class);
487            FormFieldResolvedModel fieldModel;
488            final String methodName = method.getName();
489            // ensure valid java bean getter
490            if (field != null) {
491              if (!methodName.startsWith("get")) {
492                throw new IllegalStateException("FormField annotated method must be a valid getter
                    ↪   method '" + methodName
493                    + "'");
494              }
495              fieldModel = new FormFieldResolvedModel(ctx.id, methodName, field);
496              models.add(fieldModel);
497            }
```

```
498        }
499
500        // additional annotations
501        for (Method method : methods) {
502
503          final SelectFormField select = method.getAnnotation(SelectFormField.class);
504          FormFieldResolvedModel fieldModel;
505
506          if (select != null) {
507            int index = -1;
508            final String target = select.target();
509            final String targetGetter = new StringBuilder("get").append(target.substring(0, 1)
510                                          .toUpperCase())
511                                        .append(target.substring(1, target.length()))
512                                        .toString();
513            if ((index = models.indexOf(new FormFieldResolvedModel(ctx.id, targetGetter))) == -1) {
514              throw new IllegalStateException("SelectFormField target '" + targetGetter + "' field
                ↪   model '" + model.getClass()
515                                                        .getName() + "' not found");
516            }
517            fieldModel = models.get(index);
518            try {
519              final ObservableList<Object> dataList = (ObservableList<Object>) method.invoke(model);
520              fieldModel.putData(fieldModel.id, dataList);
521              fieldModel.putData(select.annotationType(), select);
522            } catch (Throwable e) {
523              throw new IllegalStateException("Cannot retrieve select data (? extends Iterable<?>)",
                ↪   e);
524            }
525          }
526        }
527
528        // Sort models by annotation provided ordinal
529        Collections.sort(models, new Comparator<FormFieldResolvedModel>() {
530          @Override
531          public int compare(FormFieldResolvedModel o1, FormFieldResolvedModel o2) {
532            return Integer.valueOf(o1.field.ordinal())
533                    .compareTo(o2.field.ordinal());
534          }
535        });
536        return models;
537    }
538 }
```

1.6 Source-Code(Implementation Data Model)

Folgend ist der Source der Daten Modelle angeführt.
Dieses Model kann dann in weiterer Folge zu JPA Entitäten umgewandelt werden.

1.6.1 AbstractEntity.java

Folgende abstrakte Klasse stellt die root Entität für alle Entitäten dar und implementiert bereits `_getId(); setId();`, wobei `_getId()` dafür getdacht ist um die private Datenkomponente id zu erhalten sondern um die Subklassen dazu zu zwingen, dass sie `getId()` überschreiben und ein `@Id` Mapping definieren. (JPA relevant)

Listing 15: AbstractEntity.java

```java
package at.fh.ooe.swe4.fx.campina.jpa.api;

import java.io.Serializable;

public abstract class AbstractEntity<I extends Serializable> implements Serializable {

  private static final long  serialVersionUID  = 1095329951571671581L;

  private I         id;

  public AbstractEntity() {
  }

  public AbstractEntity(I id) {
    super();
    this.id = id;
  }

  public abstract I getId();

  protected I _getId() {
    return id;
  }

  public void setId(I id) {
    this.id = id;
  }

  @Override
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((id == null) ? 0 : id.hashCode());
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    AbstractEntity<I> other = (AbstractEntity<I>) obj;
    if (id == null) {
      if (other.id != null)
        return false;
```

```
49      } else if (!id.equals(other.id))
50        return false;
51      return true;
52    }
53
54 }
```

1.6.2 Day.java

Folgende Enumeration spezifiziert die zuer Verfügung stehenden Tage und auch einen Label, der in einen Produktivsystem über Keys in Form von String, Enumeration abgebildet werden sollte um Internationalisierung zu realisieren.

Dieser Datentyp kann in JPA mit Hibernate nativ gemapped werden ansonsten müsste man die Enumeration in einen String `enum.name()` serialisieren und wieder zu einer Emuneration de-serialisieren,w as sich aber als nicht schwierig herausstellen sollte.

Listing 16: Day.java

```java
package at.fh.ooe.swe4.fx.campina.jpa.constants;

import java.util.Objects;

/**
 * Enumeration which specifies the available days.
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public enum Day {
  MONDAY("Montag"),
  THUESDAY("Dienstag"),
  WEDNESDAY("Mittwoch"),
  THURSDAY("Donnerstag"),
  FRIDAY("Freitag"),
  SATARDAY("Samstag"),
  SUNDAY("Sontag");

  public final String  label;

  private Day(String label) {
    this.label = label;
  }

  /**
   * Resolve {@link Day} by its held label.
   *
   * @param label
   *            the label of the enumeration
   * @return the found {@link Day} instance
   * @throws NullPointerException
   *             if the label is null
   */
  public static final Day fromLabel(final String label) {
    Objects.requireNonNull(label);

    for (Day day : Day.values()) {
      if (day.label.equals(label)) {
        return day;
      }
    }
    throw new IllegalArgumentException("No day with label '" + label + "' found");
  }
}
```

1.6.3 User.java

Folgende Klasse stellt den User auf der Datenbank dar.

Listing 17: User.java

```java
package at.fh.ooe.swe4.fx.campina.jpa;

import java.util.HashSet;
import java.util.Set;

import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;

/**
 * The campina user.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class User extends AbstractEntity<Integer> {

  private static final long  serialVersionUID  = 94057306870207307L;

  private String        firstName;
  private String        lastName;
  private String        email;
  private String        username;
  private String        password;
  private Boolean         adminFlag      = Boolean.FALSE;
  private Boolean         blockedFlag     = Boolean.FALSE;
  private Set<Order>      orders        = new HashSet<>();

  /**
   *
   */
  public User() {
  }

  /**
   * @param id
   * @param firstName
   * @param lastName
   * @param email
   */
  public User(Integer id, String firstName, String lastName, String email) {
    super(id);
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
  }

  public Integer getId() {
    return _getId();
  }

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
```

```
59        return lastName;
60      }
61
62      public void setLastName(String lastName) {
63        this.lastName = lastName;
64      }
65
66      public String getEmail() {
67        return email;
68      }
69
70      public void setEmail(String email) {
71        this.email = email;
72      }
73
74      public String getUsername() {
75        return username;
76      }
77
78      public void setUsername(String username) {
79        this.username = username;
80      }
81
82      public String getPassword() {
83        return password;
84      }
85
86      public void setPassword(String password) {
87        this.password = password;
88      }
89
90      public Boolean getAdminFlag() {
91        return adminFlag;
92      }
93
94      public void setAdminFlag(Boolean adminFlag) {
95        this.adminFlag = adminFlag;
96      }
97
98      public Boolean getBlockedFlag() {
99        return blockedFlag;
100     }
101
102     public void setBlockedFlag(Boolean blockedFlag) {
103       this.blockedFlag = blockedFlag;
104     }
105
106     public Set<Order> getOrders() {
107       return orders;
108     }
109
110     public void setOrders(Set<Order> orders) {
111       this.orders = orders;
112     }
113
114 }
```

1.6.4 Menu.java

Folgende Klasse stellt den Menu auf der Datenbank dar.

Listing 18: Menu.java

```java
package at.fh.ooe.swe4.fx.campina.jpa;

import java.util.SortedSet;
import java.util.TreeSet;

import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;
import at.fh.ooe.swe4.fx.campina.jpa.constants.Day;

/**
 * The menu.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class Menu extends AbstractEntity<Integer> implements Comparable<Menu> {

  private static final long    serialVersionUID  = 7569033478037865818L;

  private Day              day;
  private String           label;
  private SortedSet<MenuEntry>  entires         = new TreeSet<>();

  /**
   *
   */
  public Menu() {
    super();
  }

  /**
   *
   * @param id
   */
  public Menu(Integer id) {
    super(id);
  }

  /**
   * @param id
   * @param day
   * @param label
   */
  public Menu(Integer id, Day day, String label) {
    super(id);
    this.day = day;
    this.label = label;
  }

  @Override
  public Integer getId() {
    return _getId();
  }

  public Day getDay() {
    return day;
  }

  public void setDay(Day day) {
```

```
59        this.day = day;
60    }
61
62    public String getLabel() {
63        return label;
64    }
65
66    public void setLabel(String label) {
67        this.label = label;
68    }
69
70    public SortedSet<MenuEntry> getEntries() {
71        return entires;
72    }
73
74    public void setEntires(SortedSet<MenuEntry> entires) {
75        this.entires = entires;
76    }
77
78    @Override
79    public int compareTo(Menu o) {
80        if ((getId() == null) && (o.getId() == null)) {
81            return 0;
82        } else if (getId() == null) {
83            return -1;
84        } else if (o.getId() == null) {
85            return 1;
86        } else {
87            return getId().compareTo(o.getId());
88            // return day.compareTo(o.getDay());
89        }
90    }
91 }
```

1.6.5 MenuEntry.java

Folgende Klasse stellt den Menu Eintrag auf der Datenbank dar.
Ein Menu (Fischtag) kann mehrere Gerichte haben (Forelle, Zander, ...)

Listing 19: MenuEntry.java

```java
package at.fh.ooe.swe4.fx.campina.jpa;

import java.math.BigDecimal;
import java.util.HashSet;
import java.util.Set;

import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;

/**
 * The menu entry
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class MenuEntry extends AbstractEntity<Integer> implements Comparable<MenuEntry> {

  private static final long  serialVersionUID  = -1771596592654083392L;

  private Integer       ordinal;
  private String        label;
  private BigDecimal      price;
  private Menu        menu;
  private Set<Order>      orders       = new HashSet<>();

  /**
   *
   */
  public MenuEntry() {
    super();
  }

  /**
   * @param id
   */
  public MenuEntry(Integer id) {
    super(id);
  }

  /**
   * @param id
   * @param ordinal
   * @param label
   * @param price
   * @param menu
   */
  public MenuEntry(Integer id, Integer ordinal, String label, BigDecimal price, Menu menu) {
    this(id);
    this.ordinal = ordinal;
    this.label = label;
    this.price = price;
    this.menu = menu;
  }

  @Override
  public Integer getId() {
    return _getId();
```

```
57      }
58
59      public Integer getOrdinal() {
60          return ordinal;
61      }
62
63      public void setOrdinal(Integer ordinal) {
64          this.ordinal = ordinal;
65      }
66
67      public String getLabel() {
68          return label;
69      }
70
71      public void setLabel(String label) {
72          this.label = label;
73      }
74
75      public BigDecimal getPrice() {
76          return price;
77      }
78
79      public void setPrice(BigDecimal price) {
80          this.price = price;
81      }
82
83      public Menu getMenu() {
84          return menu;
85      }
86
87      public void setMenu(Menu menu) {
88          this.menu = menu;
89      }
90
91      public Set<Order> getOrders() {
92          return orders;
93      }
94
95      public void setOrders(Set<Order> orders) {
96          this.orders = orders;
97      }
98
99      @Override
100     public int compareTo(MenuEntry o) {
101         if ((getId() == null) && (o.getId() == null)) {
102             return 0;
103         } else if (getId() == null) {
104             return -1;
105         } else if (o.getId() == null) {
106             return 1;
107         } else {
108             return getId().compareTo(o.getId());
109             // return day.compareTo(o.getDay());
110         }
111     }
112 }
```

1.6.6 Order.java

Folgende Klasse stellt den Order Eintrag auf der Datenbank dar.
Ein Menu (Fischtag) kann mehrere Gerichte haben (Forelle, Zander, ...)

Listing 20: Order.java

```java
package at.fh.ooe.swe4.fx.campina.jpa;

import java.util.Calendar;

import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;

/**
 * The order on the database.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class Order extends AbstractEntity<Integer> {

  private static final long  serialVersionUID  = -4218683353334647328L;

  private User         user;
  private MenuEntry      menuEntry;
  private Calendar       orderDate;
  private Calendar       collectDate;

  public Order(Integer id, User user, MenuEntry menuEntry, Calendar orderDate, Calendar
    ↪   collectDate) {
    super(id);
    this.user = user;
    this.menuEntry = menuEntry;
    this.orderDate = orderDate;
    this.collectDate = collectDate;
  }

  /**
   *
   */
  public Order() {
  }

  /**
   * @param id
   */
  public Order(Integer id) {
    super(id);
  }

  @Override
  public Integer getId() {
    return _getId();
  }

  public User getUser() {
    return user;
  }

  public void setUser(User user) {
    this.user = user;
  }

```

```java
public MenuEntry getMenuEntry() {
  return menuEntry;
}

public void setMenuEntry(MenuEntry menuEntry) {
  this.menuEntry = menuEntry;
}

public Calendar getOrderDate() {
  return orderDate;
}

public void setOrderDate(Calendar orderDate) {
  this.orderDate = orderDate;
}

public Calendar getCollectDate() {
  return collectDate;
}

public void setCollectDate(Calendar collectDate) {
  this.collectDate = collectDate;
}
}
```

1.6.7 LoginEvent.java

Folgende Klasse stellt den Login Eintrag auf der Datenbank dar.
Ermöglicht die Nachverfolgbarkeit der Logins der Benutzer.

Listing 21: LoginEvent.java

```java
package at.fh.ooe.swe4.fx.campina.jpa;

import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;

/**
 * The login event which represents an valid login maybe an invalid too.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class LoginEvent extends AbstractEntity<Integer> {

  private static final long  serialVersionUID  = 1782498743658504195L;

  // TODO: Add user references and calendar fields for login date

  /**
   *
   */
  public LoginEvent() {
    this(null);
  }

  /**
   * @param id
   */
  public LoginEvent(Integer id) {
    super(id);
  }

  @Override
  public Integer getId() {
    return _getId();
  }
}
```

1.6.8 EntityCache.java

Folgende Klasse dient zu Simulation einer Datenbank damit Daten innerhalb der Laufzeit persistent gehalten werden.

Listing 22: EntityCache.java

```java
package at.fh.ooe.swe4.fx.campina.jpa;

import java.io.Serializable;
import java.math.BigDecimal;
import java.util.Calendar;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Objects;
import java.util.Set;

import at.fh.ooe.swe4.fx.campina.jpa.constants.Day;
import at.fh.ooe.swe4.fx.campina.view.admin.login.model.LoginModel;

public class EntityCache {

  public static Set<User>       userCache      = new HashSet<>(100);
  public static Set<Menu>       menuCache      = new HashSet<>(100);
  public static Set<MenuEntry>  menuEntryCache  = new HashSet<>(100);
  public static Set<Order>      orderCache      = new HashSet<>(100);

  static {
    final User u1 = new User(1, "Thomas", "Herzog", "t.herzog@bla.bla");
    final User u2 = new User(2, "Hugo", "Fichtner", "h.fichtner@bla.bla");
    final User u3 = new User(3, "Christian", "Beikov", "c.beickov@bla.bla");
    final User u4 = new User(4, "Rainer", "Rudolf", "r.rudolf@bla.bla");
    final User u5 = new User(5, "Bernd", "Maier", "b.maier@bla.bla");

    final Menu m1 = new Menu(1, Day.MONDAY, "Fisch Tag");
    final Menu m2 = new Menu(1, Day.WEDNESDAY, "Steak Tag");
    final Menu m3 = new Menu(1, Day.THURSDAY, "Nudel Tag");

    final MenuEntry me1 = new MenuEntry(1, 1, "Forelle", BigDecimal.ONE, m1);
    final MenuEntry me2 = new MenuEntry(2, 2, "Zander", BigDecimal.ONE, m1);
    m1.getEntries()
      .add(me1);
    m1.getEntries()
      .add(me2);

    final Order o1 = new Order(1, u1, me1, Calendar.getInstance(), Calendar.getInstance());
    me1.getOrders()
      .add(o1);
    u1.getOrders()
      .add(o1);
    final Order o2 = new Order(2, u2, me2, Calendar.getInstance(), Calendar.getInstance());
    me1.getOrders()
      .add(o1);
    u2.getOrders()
      .add(o1);

    userCache.add(u1);
    userCache.add(u2);
    userCache.add(u3);
    userCache.add(u4);
    userCache.add(u5);

    menuEntryCache.add(me1);
```

```java
57        menuEntryCache.add(me2);
58
59        menuCache.add(m1);
60        menuCache.add(m2);
61        menuCache.add(m3);
62
63        orderCache.add(o1);
64        orderCache.add(o2);
65    }
66
67    private EntityCache() {
68    }
69
70    public static final User isValidLogin(final LoginModel model) {
71        Objects.requireNonNull(model);
72
73        final Iterator<User> it = userCache.iterator();
74        boolean valid = Boolean.FALSE;
75        User user = null;
76        while ((it.hasNext()) && (!valid)) {
77            user = it.next();
78            valid = ((model.getUsername().equals(user.getUsername())) &&
             ↪    (model.getPassword().equals(user.getPassword())));
79        }
80
81        return valid ? user : null;
82    }
83
84    public static final Menu byMenuId(final Serializable id) {
85        Objects.requireNonNull(id);
86
87        for (Menu menu : menuCache) {
88            if (id.equals(menu.getId())) {
89                return menu;
90            }
91        }
92        throw new IllegalArgumentException("id does not correspond to an menu");
93    }
94
95    public static final void deleteForMenuId(final Serializable id) {
96        Objects.requireNonNull(id);
97
98        final Iterator<MenuEntry> it = menuEntryCache.iterator();
99        while (it.hasNext()) {
100           final MenuEntry entry = it.next();
101           if (entry.getId()
102                   .equals(id)) {
103               it.remove();
104           }
105       }
106   }
107 }
```

1.7 Source-Code(Implementation Login View)

Folgend ist der Source der Login View angeführt.

1.7.1 LoginModel.java

Folgend ist das Login Model angeführt welches für das Login Formular verwendet wird.

Listing 23: LoginModel.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.login.model;

import java.util.Objects;

import at.fh.ooe.swe4.fx.campina.jpa.LoginEvent;
import at.fh.ooe.swe4.fx.campina.view.annotation.FormField;
import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;

/**
 * The model backing the {@link LoginEvent} entity.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class LoginModel extends AbstractViewModel<Integer, LoginEvent> {

  private Integer        counter                = 0;
  private String         username               = "hello";
  private String         password               = "hello";

  public static final Integer  MAX_COUNTER                 = Integer.valueOf(5);
  public static final String   LOGIN_GREETING_MESSAGE_PATTERN  = "Sie haben %d Versuche um sich
    ↪  einzuloggen";
  public static final String   LOGIN_FAILED_MESSAGE_PATTERN   = "Login ungültig !!! %d Versuche
    ↪  verbleiben";
  public static final String   LOGIN_BLOCKED_MESSAGE       = "Zu viele Fehlversuche !!! Zugang
    ↪  gesperrt";

  @Override
  public void reset() {
    prepare(new LoginEvent());
  }

  @Override
  public void prepare(LoginEvent loginEvent) {
    Objects.requireNonNull(loginEvent);

    setEntity(loginEvent);
    setId(loginEvent.getId());
  }

  public void increaseCounter() {
    counter++;
  }

  @FormField(
      id = "login-username",
      label = "Benutzernamen",
      ordinal = 1,
      required = true,
      requiredMessage = "Bitte Benutzernamen angeben",
      type = FormFieldType.INPUT_TEXT)
```

```
51    public String getUsername() {
52      return username;
53    }
54
55    public void setUsername(String username) {
56      this.username = username;
57    }
58
59    @FormField(
60        id = "login-password",
61        label = "Passwort",
62        ordinal = 1,
63        required = true,
64        requiredMessage = "Bitte Passwort angeben",
65        type = FormFieldType.INPUT_TEXT)
66    public String getPassword() {
67      return password;
68    }
69
70    public void setPassword(String password) {
71      this.password = password;
72    }
73
74    public Integer getCounter() {
75      return counter;
76    }
77
78  }
```

1.7.2 LoginEventControl.java

Folgend ist die Klasse angeführt, welche die Actions behandelt, wie Speichern, Löschen, usw.

Listing 24: LoginEventControl.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.login.control;

import javafx.event.ActionEvent;
import javafx.scene.Node;
import javafx.scene.control.Button;
import javafx.scene.text.Text;
import javafx.scene.text.TextFlow;

import org.apache.commons.lang.StringUtils;

import at.fh.ooe.swe4.fx.campina.jpa.EntityCache;
import at.fh.ooe.swe4.fx.campina.jpa.User;
import at.fh.ooe.swe4.fx.campina.view.admin.login.model.LoginModel;
import at.fh.ooe.swe4.fx.campina.view.admin.login.part.LoginTabViewHandler;
import at.fh.ooe.swe4.fx.campina.view.api.FormContext;

/**
 * This class contains all of the action logic:
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class LoginEventControl {

  private User  loggedUser  = null;

  /**
   *
   */
  public LoginEventControl() {
  }

  /**
   * Handles the login event
   *
   * @param eventthe
   *            {@link ActionEvent}
   */
  public void handleLogin(final ActionEvent event) {
    final FormContext<LoginModel> ctx = (FormContext<LoginModel>) ((Node)
        ↪  event.getSource()).getUserData();
    populateFormMessage(null, ctx);
    ctx.formHandler.validateForm(ctx);

    if (ctx.valid) {
      ctx.formHandler.fillModel(ctx);

      // TODO: Search for username and password on db
      loggedUser = EntityCache.isValidLogin(ctx.model);
      // increase counter
      if (loggedUser == null) {
        ctx.model.increaseCounter();
        if (ctx.model.getCounter() >= LoginModel.MAX_COUNTER) {
          populateFormMessage(LoginModel.LOGIN_BLOCKED_MESSAGE, ctx);
          ((Button) ctx.getNode(LoginTabViewHandler.LOGIN_BUTTON_ID)).setDisable(Boolean.TRUE);
        } else {
          populateFormMessage(String.format(LoginModel.LOGIN_FAILED_MESSAGE_PATTERN,
              ↪  (LoginModel.MAX_COUNTER - ctx.model.getCounter())), ctx);
```

```
57              }
58          }
59      }
60      // for invalid
61      else {
62          populateFormMessage("Formular ungültig !!! Bitte Eingaben prüfen", ctx);
63      }
64  }
65
66  /**
67   * Populates a message to the message box. <br>
68   * If message is null the actual set message will be cleared
69   *
70   * @param message
71   *            the message to populate
72   * @param ctx
73   *            the form context
74   */
75  private void populateFormMessage(final String message, final FormContext<LoginModel> ctx) {
76      final TextFlow flow = ((TextFlow) ctx.getNode(LoginTabViewHandler.FORM_MESSAGE));
77      flow.getChildren()
78          .clear();
79      flow.setPrefHeight(0);
80      if (!StringUtils.isEmpty(message)) {
81          flow.getChildren()
82              .add(new Text(message));
83          flow.setPrefHeight(30);
84      }
85  }
86 }
```

1.7.3 LoginTabViewHandler.java

Folgend ist der ViewHanddler der LoginTab angeführt. Sie stellt einen View Teil der Scene dar und implementiert `ViewHandler<T>` Interface.

Listing 25: LoginTabViewHandler.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.login.part;

import java.util.Objects;

import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Tab;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Text;
import javafx.scene.text.TextFlow;
import at.fh.ooe.swe4.fx.campina.view.admin.login.control.LoginEventControl;
import at.fh.ooe.swe4.fx.campina.view.admin.login.model.LoginModel;
import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
import at.fh.ooe.swe4.fx.campina.view.api.ViewHandler;
import at.fh.ooe.swe4.fx.campina.view.form.FormHandler;

/**
 * {@link ViewHandler} implementation for the login tab.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class LoginTabViewHandler implements ViewHandler<Tab> {

  /**
   * The form builder for the user form
   */
  private final FormHandler<LoginModel>  formHandler;
  /**
   * The user form controller
   */
  private final LoginEventControl       loginControl;
  /**
   * The form context for the user form
   */
  final FormContext<LoginModel>       ctx;

  // ############################################################################
  // Ids of nodes and form backed model class
  // ############################################################################
  public static final String          LOGIN_FORM_ID  = "login-form";
  public static final String          FORM_MESSAGE   = "login-form-message";
  public static final String          LOGIN_BUTTON_ID  = "login-button-login";

  /**
   * @param scene
   */
  public LoginTabViewHandler(final Scene scene) {
    Objects.requireNonNull(scene);

    final LoginModel model = new LoginModel();
    model.reset();
    this.formHandler = new FormHandler<LoginModel>().init();
    this.loginControl = new LoginEventControl();
    this.ctx = new FormContext<LoginModel>(LOGIN_FORM_ID, formHandler, model, scene);
```

```java
57      }
58
59      // ########################################################################
60      // ScenePart methods
61      // ########################################################################
62      @Override
63      public String getId() {
64        return "login-tab";
65      }
66
67      @Override
68      public Tab createNode() {
69        // Form message text
70        final TextFlow flow = new TextFlow();
71        flow.setId(FORM_MESSAGE);
72        flow.getChildren()
73          .add(new Text(String.format(LoginModel.LOGIN_GREETING_MESSAGE_PATTERN,
            ↪   LoginModel.MAX_COUNTER)));
74        flow.setStyle("-fx-font-size: 20pt");
75        ctx.putNode(FORM_MESSAGE, flow);
76
77        // Form
78        final Node form = formHandler.generateFormGrid(ctx);
79
80        // Buttons
81        final Button loginButton = new Button();
82        loginButton.setId(LOGIN_BUTTON_ID);
83        loginButton.setText("Login");
84        loginButton.setUserData(ctx);
85        loginButton.setOnAction(loginControl::handleLogin);
86        ctx.putNode(LOGIN_BUTTON_ID, loginButton);
87
88        final GridPane pane = new GridPane();
89        pane.setId("user-tab-content");
90
91        // left part of tab
92        final GridPane mainGrid = new GridPane();
93        mainGrid.setId("login-form-container");
94        mainGrid.add(flow, 0, 0);
95        mainGrid.add(form, 0, 1);
96        mainGrid.add(loginButton, 0, 2);
97        mainGrid.setPrefHeight(500);
98        pane.add(mainGrid, 0, 0);
99
100       final Tab tab = new Tab(getId());
101       tab.setText("Login");
102       tab.setClosable(false);
103       tab.setContent(pane);
104
105       return tab;
106     }
107
108     @Override
109     public void initHandler() {
110       ctx.formHandler.fillForm(ctx);
111     }
112   }
```

1.8 Source-Code(Implementation User View)

Folgend ist der Source der User View angeführt.

1.8.1 UserModel.java

Folgend ist das User Model angeführt welches für das User Formular verwendet wird.

Listing 26: UserModel.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.user.model;

import java.util.Objects;

import at.fh.ooe.swe4.fx.campina.jpa.User;
import at.fh.ooe.swe4.fx.campina.view.annotation.FormField;
import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;

/**
 * The view model which backs the {@link User} entity.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class UserModel extends AbstractViewModel<Integer, User> {

  /**
   *
   */
  public UserModel() {
    super();
  }

  /**
   * @param user
   */
  public UserModel(User user) {
    super(user);
  }

  @Override
  public void prepare(User user) {
    Objects.requireNonNull(user);

    setId(user.getId());
    setEntity(user);
  }

  @Override
  public void reset() {
    prepare(new User());
  }

  public String getSelectionName() {
    if (getId() == null) {
      return "Neuer Benutzer";
    } else {
      return new StringBuilder().append(getLastName())
                    .append(", ")
                    .append(getFirstName())
                    .append(" (")
                    .append(getEmail())
```

```
54                    .append(")")
55                    .toString();
56        }
57    }
58
59    @FormField(
60        id = "user-first-name",
61        label = "Vorname",
62        required = true,
63        ordinal = 1,
64        requiredMessage = "Bitte Vornamen angeben",
65        type = FormFieldType.INPUT_TEXT)
66    public String getFirstName() {
67      return getEntity().getFirstName();
68    }
69
70    public void setFirstName(String firstName) {
71      getEntity().setFirstName(firstName);
72      ;
73    }
74
75    @FormField(
76        id = "user-last-name",
77        label = "Nachname",
78        required = true,
79        ordinal = 2,
80        requiredMessage = "Bitte Nachnamen angeben",
81        type = FormFieldType.INPUT_TEXT)
82    public String getLastName() {
83      return getEntity().getLastName();
84    }
85
86    public void setLastName(String lastName) {
87      getEntity().setLastName(lastName);
88      ;
89    }
90
91    @FormField(
92        id = "user-email",
93        label = "E-Mail",
94        ordinal = 3,
95        type = FormFieldType.INPUT_TEXT)
96    public String getEmail() {
97      return getEntity().getEmail();
98    }
99
100   public void setEmail(String email) {
101     getEntity().setEmail(email);
102   }
103
104   @FormField(
105       id = "user-username",
106       label = "Benutzername",
107       required = true,
108       ordinal = 4,
109       requiredMessage = "Bitte Benutzernamen angeben",
110       type = FormFieldType.INPUT_TEXT)
111   public String getUsername() {
112     return getEntity().getUsername();
113   }
114
115   public void setUsername(String username) {
116     getEntity().setUsername(username);
```

```
117      }
118
119      @FormField(
120          id = "user-password",
121          label = "Password",
122          required = true,
123          ordinal = 5,
124          requiredMessage = "Bitte Password angeben",
125          type = FormFieldType.INPUT_TEXT)
126      public String getPassword() {
127        return getEntity().getPassword();
128      }
129
130      public void setPassword(String password) {
131        getEntity().setPassword(password);
132      }
133
134  }
```

1.8.2 UserEventControl.java

Folgend ist die Klasse angeführt, welche die Actions behandelt, wie Speichern, Löschen, usw.

Listing 27: UserEventControl.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.user.control;

import java.util.Objects;

import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.scene.Node;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.text.Text;
import javafx.scene.text.TextFlow;

import org.apache.commons.lang.StringUtils;

import at.fh.ooe.swe4.fx.campina.jpa.EntityCache;
import at.fh.ooe.swe4.fx.campina.jpa.User;
import at.fh.ooe.swe4.fx.campina.view.admin.user.model.UserModel;
import at.fh.ooe.swe4.fx.campina.view.admin.user.part.UserTabviewHandler;
import at.fh.ooe.swe4.fx.campina.view.api.FormContext;

/**
 * The control bean for the user tab.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 3, 2015
 */
public class UserEventControl {

  /**
   * Creates test data since we have no back-end yet
   */
  public UserEventControl() {
  }

  // ###########################################################
  // Button controls
  // ###########################################################
  /**
   * Handles the new action of the form.
   *
   * @param event
   *            the {@link ActionEvent}
   */
  public void handleNewAction(final ActionEvent event) {
    final FormContext<UserModel> ctx = (FormContext<UserModel>) ((Node)
      event.getSource()).getUserData();
    // clear former set message
    populateFormMessage(null, ctx);
    // reset form
    ctx.formHandler.resetForm(ctx);
    // create new user model with new user entity
    ctx.model.reset();
    // hide buttons
    setButtonVisibility(ctx, Boolean.FALSE);
    // reload users
    handleUserReload(ctx);
  }

```

```
58   /**
59    * Handles the save action of the form.
60    *
61    * @param event
62    *            the {@link ActionEvent}
63    */
64   public void handleSaveAction(final ActionEvent event) {
65     final FormContext<UserModel> ctx = (FormContext<UserModel>) ((Node)
        ↪   event.getSource()).getUserData();
66     // clear former set messages
67     populateFormMessage(null, ctx);
68     // validate form
69     ctx.formHandler.validateForm(ctx);
70     // is valid
71     if (ctx.valid) {
72       // fill model with form data
73       ctx.formHandler.fillModel(ctx);
74
75       // TODO: Persist entity here
76       final User user = ctx.model.getEntity();
77
78       // if not already managed increase id by size + 1
79       if (!EntityCache.userCache.contains(user)) {
80         user.setId(EntityCache.userCache.size() + 1);
81       }
82       // save model in backed list for testing
83       EntityCache.userCache.add(user);
84       // init model with new saved user
85       ctx.model.prepare(user);
86       // enable buttons
87       setButtonVisibility(ctx, Boolean.TRUE);
88       // reload data from db (now backing list)
89       handleUserReload(ctx);
90     } else {
91       populateFormMessage("Formular ungültig !!! Bitte Eingaben prüfen", ctx);
92     }
93     event.consume();
94
95   }
96
97   /**
98    * Handles the delete action of the form.
99    *
100   * @param event
101   *            the {@link ActionEvent}
102   */
103  public void handleDeleteAction(final ActionEvent event) {
104    final FormContext<UserModel> ctx = (FormContext<UserModel>) ((Node)
        ↪   event.getSource()).getUserData();
105    // clear former set message
106    populateFormMessage(null, ctx);
107    // reset the form
108    ctx.formHandler.resetForm(ctx);
109
110    final UserModel model = ((ChoiceBox<UserModel>)
        ↪   ctx.getNode(UserTabviewHandler.USER_SELECTION_KEY)).getSelectionModel()
111                                                 .getSelectedItem();
112
113    // TODO: Delete entity from db here
114
115    // existing user gets deleted
116    if (model.getId() != null) {
117      EntityCache.userCache.remove(model.getEntity());
```

```
118        }
119
120        // reset model
121        ctx.model = new UserModel();
122        // disable buttons
123        setButtonVisibility(ctx, Boolean.FALSE);
124        // reload users
125        handleUserReload(ctx);
126     }
127
128     /**
129      * Handles the block action of the form.
130      *
131      * @param event
132      *            the {@link ActionEvent}
133      */
134     public void handleBlockAction(final ActionEvent event) {
135        final FormContext<UserModel> ctx = (FormContext<UserModel>) ((Node)
            ↪   event.getSource()).getUserData();
136        // clear old set message
137        populateFormMessage(null, ctx);
138        // selected user model
139        final UserModel model = ((ChoiceBox<UserModel>)
            ↪   ctx.getNode(UserTabviewHandler.USER_SELECTION_KEY)).getSelectionModel()
140                                                  .getSelectedItem();
141
142        final Button blockButton = (Button) ctx.getNode(UserTabviewHandler.BLOCK_BUTTON_ID);
143        final User user = model.getEntity();
144
145        // invert user blocked state
146        user.setBlockedFlag(!model.getEntity()
147                    .getBlockedFlag());
148
149        ctx.model.prepare(user);
150        ctx.formHandler.fillForm(ctx);
151
152        // TODO: Update blocked flag on db
153
154        // got blocked
155        if (model.getEntity()
156              .getBlockedFlag()) {
157           blockButton.setText("Freigeben");
158        }
159        // got freed
160        else {
161           blockButton.setText("Blockieren");
162        }
163        // user reload
164        handleUserReload(ctx);
165     }
166
167     // ###########################################################
168     // Selection controls
169     // ###########################################################
170     public void handleUserSelection(final FormContext<UserModel> ctx, final UserModel user) {
171        // clear former set message
172        populateFormMessage(null, ctx);
173        // Selection present
174        if (user.getId() != null) {
175           ctx.model.prepare(user.getEntity());
176           ctx.formHandler.fillForm(ctx);
177           setButtonVisibility(ctx, Boolean.TRUE);
178        }
```

```java
179        // No selection present
180        else {
181          ctx.model.reset();
182          ctx.formHandler.fillForm(ctx);
183          setButtonVisibility(ctx, Boolean.FALSE);
184        }
185      }
186
187      // ############################################################
188      // Load controls
189      // ############################################################
190      /**
191       * Handles the load of the user for the selection
192       *
193       * @param userList
194       *            the {@link ObservableList} to add users to
195       */
196      public void handleUserReload(final FormContext<UserModel> ctx) {
197        Objects.requireNonNull(ctx);
198
199        final ObservableList<UserModel> userList = (ObservableList<UserModel>)
            ↪ ctx.getObserable(UserTabviewHandler.USER_SELECTION_KEY);
200        userList.clear();
201        userList.add(new UserModel());
202        for (User user : EntityCache.userCache) {
203          userList.add(new UserModel(user));
204        }
205
206        userList.set(userList.indexOf(ctx.model), ctx.model);
207
208        ((ChoiceBox<UserModel>)
            ↪ ctx.getNode(UserTabviewHandler.USER_SELECTION_KEY)).getSelectionModel()
209                                        .select(ctx.model);
210      }
211
212      /**
213       * Sets the button visibility of these buttons which required persistent
214       * user.
215       *
216       * @param ctx
217       *            the form context
218       * @param visible
219       *            the new visible flag
220       */
221      private void setButtonVisibility(final FormContext<UserModel> ctx, final boolean visible) {
222        Objects.requireNonNull(ctx);
223
224        ctx.getNode(UserTabviewHandler.DELETE_BUTTON_ID)
225          .setVisible(visible);
226        ctx.getNode(UserTabviewHandler.BLOCK_BUTTON_ID)
227          .setVisible(visible);
228      }
229
230      /**
231       * Populates a message to the message box. <br>
232       * If message is null the actual set message will be cleared
233       *
234       * @param message
235       *            the message to populate
236       * @param ctx
237       *            the form context
238       */
239      private void populateFormMessage(final String message, final FormContext<UserModel> ctx) {
```

```
240    final TextFlow flow = ((TextFlow) ctx.getNode(UserTabviewHandler.FORM_MESSAGE));
241    flow.getChildren()
242      .clear();
243    flow.setPrefHeight(0);
244    if (!StringUtils.isEmpty(message)) {
245      flow.getChildren()
246        .add(new Text(message));
247      flow.setPrefHeight(30);
248    }
249  }
250 }
```

1.8.3 UserTabViewHandler.java

Folgend ist der ViewHanddler der UserTab angeführt. Sie stellt einen View Teil der Scene dar und implementiert `ViewHandler<T>` Interface.

Listing 28: UserTabViewHandler.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.user.part;

import java.util.Arrays;
import java.util.Objects;

import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Tab;
import javafx.scene.layout.GridPane;
import javafx.scene.text.TextFlow;
import javafx.util.StringConverter;
import at.fh.ooe.swe4.fx.campina.view.admin.user.control.UserEventControl;
import at.fh.ooe.swe4.fx.campina.view.admin.user.model.UserModel;
import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
import at.fh.ooe.swe4.fx.campina.view.api.ViewHandler;
import at.fh.ooe.swe4.fx.campina.view.form.FormHandler;

/**
 * This class builds the user tab.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 3, 2015
 */
public class UserTabviewHandler implements ViewHandler<Tab> {

  /**
   * The form builder for the user form
   */
  private final FormHandler<UserModel>  formHandler;
  /**
   * The user form controller
   */
  private final UserEventControl      userControl;
  /**
   * The form context for the user form
   */
  final FormContext<UserModel>       ctx;

  // ############################################################################
  // Ids of nodes and form backed model class
  // ############################################################################
  public static final String        USER_SELECTION_KEY  = "user-selection";
  public static final String        SAVE_BUTTON_ID    = "user-save-button";
  public static final String        NEW_BUTTON_ID     = "user-new-button";
  public static final String        DELETE_BUTTON_ID   = "user-delete-button";
  public static final String        BLOCK_BUTTON_ID    = "user-block-button";
  public static final String        FORM_MESSAGE     = "user-form-message";

  /**
   * Inits this user tab manager for the given scene
   *
```

```java
57       * @param scene
58       *            the backing scene
59       */
60      public UserTabviewHandler(final Scene scene) {
61        Objects.requireNonNull(scene);
62
63        this.formHandler = new FormHandler<UserModel>();
64        this.formHandler.init();
65        this.userControl = new UserEventControl();
66        this.ctx = new FormContext<UserModel>("tab-user", formHandler, new UserModel(), scene);
67      }
68
69      // ############################################################################
70      // ScenePart methods
71      // ############################################################################
72      @Override
73      public String getId() {
74        return "user-admin";
75      }
76
77      @Override
78      public void initHandler() {
79        userControl.handleUserReload(ctx);
80        ((ChoiceBox<UserModel>) ctx.getNode(USER_SELECTION_KEY)).getSelectionModel()
81                                   .select(0);
82      }
83
84      @Override
85      public Tab createNode() {
86        final GridPane pane = new GridPane();
87        pane.setId(getId() + "-content");
88
89        // form
90        final GridPane formGrid = formHandler.generateFormGrid(ctx);
91
92        // left part of tab
93        final GridPane mainGrid = new GridPane();
94        mainGrid.setId("user-form");
95        mainGrid.add(createFormMessage(ctx), 0, 0);
96        mainGrid.add(createUserChoice(ctx), 0, 1);
97        mainGrid.add(formGrid, 0, 2);
98        mainGrid.add(createButtonGroup(ctx), 0, 3);
99        mainGrid.setPrefHeight(500);
100       mainGrid.setHgap(10);
101       mainGrid.setVgap(10);
102       pane.add(mainGrid, 0, 0);
103
104       final Tab tab = new Tab(getId());
105       tab.setText("Benutzerverwaltung");
106       tab.setClosable(false);
107       tab.setContent(pane);
108       return tab;
109     }
110
111     // ############################################################################
112     // Node creation methods
113     // ############################################################################
114     /**
115      * Creates the form message box where the form messages are placed
116      *
117      * @param ctx
118      *            the form context
119      * @return the {@link TextFlow} isntance for the messages
```

```
120      */
121    private TextFlow createFormMessage(final FormContext<UserModel> ctx) {
122      final TextFlow flow = new TextFlow();
123      flow.setId(getId() + "-user-form-message");
124      flow.setStyle("-fx-font-size: 20pt");
125      ctx.putNode(FORM_MESSAGE, flow);
126      return flow;
127    }
128
129    /**
130     * Creates the button group for the user tab actions.
131     *
132     * @param ctx
133     *            TODO
134     *
135     * @return a {@link GridPane} instance holding the button
136     */
137    private GridPane createButtonGroup(FormContext<UserModel> ctx) {
138      final GridPane gridPane = new GridPane();
139      gridPane.setId(getId() + "-button-grid");
140      gridPane.setHgap(10);
141      gridPane.setVgap(10);
142
143      // TODO: Register events here
144      final Button newButton = new Button();
145      newButton.setText("Zurücksetzen");
146      newButton.setId(NEW_BUTTON_ID);
147      newButton.setUserData(ctx);
148      newButton.setOnAction(userControl::handleNewAction);
149
150      final Button saveButton = new Button();
151      saveButton.setId(SAVE_BUTTON_ID);
152      saveButton.setText("Speichern");
153      saveButton.setUserData(ctx);
154      saveButton.setOnAction(userControl::handleSaveAction);
155
156      final Button deleteButton = new Button();
157      deleteButton.setId(DELETE_BUTTON_ID);
158      deleteButton.setText("Löschen");
159      deleteButton.setUserData(ctx);
160      deleteButton.setOnAction(userControl::handleDeleteAction);
161      deleteButton.setVisible(Boolean.FALSE);
162
163      final Button blockButton = new Button();
164      blockButton.setId(BLOCK_BUTTON_ID);
165      blockButton.setText("Sperren");
166      blockButton.setUserData(ctx);
167      blockButton.setOnAction(userControl::handleBlockAction);
168      blockButton.setVisible(Boolean.FALSE);
169
170      gridPane.add(newButton, 0, 0);
171      gridPane.add(saveButton, 1, 0);
172      gridPane.add(deleteButton, 2, 0);
173      gridPane.add(blockButton, 3, 0);
174
175      // register in context
176      ctx.putNode(NEW_BUTTON_ID, newButton);
177      ctx.putNode(SAVE_BUTTON_ID, saveButton);
178      ctx.putNode(DELETE_BUTTON_ID, deleteButton);
179      ctx.putNode(BLOCK_BUTTON_ID, blockButton);
180
181      return gridPane;
182    }
```

```
183
184    /**
185     * Creates the user choice node for the user selection.
186     *
187     * @param ctxthe
188     *            form context
189     * @return the {@link ChoiceBox} instance
190     */
191    private ChoiceBox<UserModel> createUserChoice(final FormContext<UserModel> ctx) {
192      Objects.requireNonNull(ctx);
193
194      final ObservableList<UserModel> users = FXCollections.observableArrayList(Arrays.asList(new
       ↪  UserModel()));
195
196      final ChoiceBox<UserModel> userChoice = new ChoiceBox<>(users);
197      userChoice.setUserData(ctx);
198      userChoice.getSelectionModel()
199            .selectedItemProperty()
200            .addListener(new ChangeListener<UserModel>() {
201              @Override
202              public void changed(ObservableValue<? extends UserModel> observable, UserModel
                ↪  oldValue, UserModel newValue) {
203                if (observable.getValue() == null) {
204                  return;
205                }
206
207                userControl.handleUserSelection(ctx, observable.getValue());
208              }
209            });
210      userChoice.setConverter(new StringConverter<UserModel>() {
211
212        @Override
213        public String toString(UserModel object) {
214          return object.getSelectionName();
215        }
216
217        @Override
218        public UserModel fromString(String string) {
219          throw new UnsupportedOperationException("Not supported conversion from string to object");
220        }
221      });
222      userChoice.setPrefWidth(400);
223      ctx.putObserable(USER_SELECTION_KEY, users);
224      ctx.putNode(USER_SELECTION_KEY, userChoice);
225      return userChoice;
226    }
227  }
```

1.9 Source-Code(Implementation Menu View)

Folgend ist der Source der Menu View angeführt.

1.9.1 MenuModel.java

Folgend ist das Menu Model angeführt welches für das Menu Formular verwendet wird.

Listing 29: MenuModel.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.menu.model;

import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.SortedSet;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.util.StringConverter;
import at.fh.ooe.swe4.fx.campina.jpa.Menu;
import at.fh.ooe.swe4.fx.campina.jpa.MenuEntry;
import at.fh.ooe.swe4.fx.campina.jpa.constants.Day;
import at.fh.ooe.swe4.fx.campina.view.annotation.FormField;
import at.fh.ooe.swe4.fx.campina.view.annotation.SelectFormField;
import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;

/**
 * The view model which backs the {@link Menu} entity.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class MenuModel extends AbstractViewModel<Integer, Menu> {

  private static final ObservableList<Day>  days  = FXCollections.observableArrayList();

  static {
    days.add((Day) null);
    days.addAll(Arrays.asList(Day.values()));
  }

  /**
   * The converter for the {@link Day} type.
   *
   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
   * @date Jun 5, 2015
   */
  public static class DayConverter extends StringConverter<Day> {

    private static final String  PLEASE_CHOOSE  = "Bitte wählen";

    public DayConverter() {
      super();
    }

    @Override
    public String toString(Day object) {
      return (object == null) ? PLEASE_CHOOSE : object.label;
    }

    @Override
```

```java
54    public Day fromString(String string) {
55      return (PLEASE_CHOOSE.equals(string)) ? null : Day.fromLabel(string);
56    }
57
58  }
59
60  /**
61   * The converter fro the {@link MenuModel} type.
62   *
63   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
64   * @date Jun 5, 2015
65   */
66  public static class MenuModelConverter extends StringConverter<MenuModel> {
67
68    @Override
69    public String toString(MenuModel object) {
70      Objects.requireNonNull(object);
71      return (object.getId() == null) ? "Neues Menu" : object.getLabel();
72    }
73
74    @Override
75    public MenuModel fromString(String string) {
76      throw new UnsupportedOperationException("Form text not supported by this converter");
77    }
78
79  }
80
81  /**
82   *
83   */
84  public MenuModel() {
85    super();
86  }
87
88  /**
89   * @param entity
90   */
91  public MenuModel(Menu entity) {
92    super(entity);
93  }
94
95  @Override
96  public void reset() {
97    prepare(new Menu());
98  }
99
100 @Override
101 public void prepare(Menu entity) {
102   super.prepare(entity);
103   entity.setDay(days.get(days.indexOf(entity.getDay())));
104 }
105
106 @FormField(id = "menu-label",
107     label = "Label",
108     ordinal = 1,
109     required = true,
110     requiredMessage = "Bitte Menu Label angeben",
111     type = FormFieldType.INPUT_TEXT)
112 public String getLabel() {
113   return getEntity().getLabel();
114 }
115
116 public void setLabel(String label) {
```

```
117        getEntity().setLabel(label);
118    }
119
120    @FormField(id = "menu-day",
121        label = "Tag",
122        ordinal = 6,
123        required = true,
124        requiredMessage = "Bitte Tag wählen",
125        type = FormFieldType.SELECT,
126        valueClass = Day.class)
127    public Day getDay() {
128      return getEntity().getDay();
129    }
130
131    public void setDay(Day day) {
132      getEntity().setDay(day);
133    }
134
135    public void setLabel(Day day) {
136      System.out.println(day);
137    }
138
139    @SelectFormField(target = "day",
140        converter = DayConverter.class)
141    public List<Day> getDays() {
142      return days;
143    }
144
145    public SortedSet<MenuEntry> getMenuEntries() {
146      return getEntity().getEntries();
147    }
148
149 }
```

1.9.2 MenuEventControl.java

Folgend ist die Klasse angeführt, welche die Actions behandelt, wie Speichern, Löschen, usw.

Listing 30: MenuEventControl.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.menu.control;

import java.util.Objects;

import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.scene.Node;
import javafx.scene.control.ChoiceBox;
import javafx.scene.text.Text;
import javafx.scene.text.TextFlow;

import org.apache.commons.lang.StringUtils;

import at.fh.ooe.swe4.fx.campina.jpa.EntityCache;
import at.fh.ooe.swe4.fx.campina.jpa.Menu;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuModel;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.part.MenuTabViewHandler;
import at.fh.ooe.swe4.fx.campina.view.admin.user.model.UserModel;
import at.fh.ooe.swe4.fx.campina.view.api.FormContext;

/**
 * The event control for the {@link Menu} entity.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class MenuEventControl {

  public MenuEventControl() {
    // TODO Auto-generated constructor stub
  }

  /**
   * Handles the new action of the form.
   *
   * @param event
   *            the {@link ActionEvent}
   */
  public void newAction(final ActionEvent event) {
    final FormContext<MenuModel> ctx = (FormContext<MenuModel>) ((Node)
      event.getSource()).getUserData();
    // clear former set message
    populateFormMessage(null, ctx);
    // reset form
    ctx.formHandler.resetForm(ctx);
    // create new user model with new user entity
    ctx.model.reset();
    // hide buttons
    ctx.getNode(MenuTabViewHandler.MENU_DELETE_BUTTON_ID)
      .setVisible(Boolean.FALSE);
    // reload users
    handleMenuReload(ctx);
  }

  /**
   * Saves a menu
   *
   * @param event
```

```
58        *            the {@link ActionEvent}
59        */
60     public void saveMenu(final ActionEvent event) {
61        final FormContext<MenuModel> ctx = (FormContext<MenuModel>) ((Node)
            ↪   event.getSource()).getUserData();
62        ctx.getNode(MenuTabViewHandler.MENU_DELETE_BUTTON_ID)
63          .setVisible(Boolean.TRUE);
64        ctx.formHandler.validateForm(ctx);
65        ctx.formHandler.fillModel(ctx);
66
67        if (ctx.valid) {
68          final Menu menu;
69          if (ctx.model.getId() != null) {
70            menu = ctx.model.getEntity();
71            EntityCache.menuCache.add(menu);
72          } else {
73            menu = ctx.model.getEntity();
74            menu.setId(EntityCache.menuCache.size() + 1);
75            EntityCache.menuCache.add(menu);
76          }
77          ctx.model.prepare(menu);
78          handleMenuReload(ctx);
79        } else {
80          populateFormMessage("Formular ungültig !! Bitte Eingaben prüfen", ctx);
81          handleMenuReload(ctx);
82        }
83     }
84
85     /**
86      * Deletes an menu
87      *
88      * @param event
89      *            the {@link ActionEvent}
90      */
91     public void deleteMenu(final ActionEvent event) {
92        final FormContext<MenuModel> ctx = (FormContext<MenuModel>) ((Node)
            ↪   event.getSource()).getUserData();
93
94        if (ctx.model.getId() != null) {
95          ctx.getNode(MenuTabViewHandler.MENU_DELETE_BUTTON_ID)
96            .setVisible(Boolean.TRUE);
97          final Menu menu = ctx.model.getEntity();
98          EntityCache.menuCache.remove(menu);
99          EntityCache.deleteForMenuId(ctx.model.getEntity()
100                           .getId());
101         ctx.formHandler.resetForm(ctx);
102         ctx.model.reset();
103       }
104       handleMenuReload(ctx);
105
106    }
107
108    /**
109     * The handles the load of the menus. Resets the current ctx.model in the
110     * observed list and set this model as selected
111     *
112     * @param ctx
113     *            the form context
114     */
115    public void handleMenuReload(final FormContext<MenuModel> ctx) {
116       Objects.requireNonNull(ctx);
117
118       final ObservableList<MenuModel> list = (ObservableList<MenuModel>)
           ↪   ctx.getObserable(MenuTabViewHandler.MENU_SELECTION_KEY);
```

```
119      list.clear();
120      list.add(new MenuModel());
121      for (Menu menu : EntityCache.menuCache) {
122        final MenuModel model = new MenuModel();
123        model.prepare(menu);
124        list.add(model);
125      }
126
127      // need to replace observed instance
128      list.set(list.indexOf(ctx.model), ctx.model);
129
130      // need to select current context hold model
131      ((ChoiceBox<MenuModel>)
        ↪   ctx.getNode(MenuTabViewHandler.MENU_SELECTION_KEY)).getSelectionModel()
132                                    .select(ctx.model);
133    }
134
135    /**
136     * Populates a message to the message box. <br>
137     * If message is null the actual set message will be cleared
138     *
139     * @param message
140     *            the message to populate
141     * @param ctx
142     *            the form context
143     */
144    private void populateFormMessage(final String message, final FormContext<MenuModel> ctx) {
145      final TextFlow flow = ((TextFlow) ctx.getNode(MenuTabViewHandler.MENU_FORM_MESSAGE));
146      flow.getChildren()
147        .clear();
148      flow.setPrefHeight(0);
149      if (!StringUtils.isEmpty(message)) {
150        flow.getChildren()
151          .add(new Text(message));
152        flow.setPrefHeight(30);
153      }
154    }
155 }
```

1.9.3 MenuEntryModel.java

Folgend ist das Menu-Eintrag Model angeführt welches für das Menu-Eintrag Formular verwendet wird.

Listing 31: MenuEntryModel.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.menu.model;

import java.math.BigDecimal;
import java.util.Objects;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.util.StringConverter;
import at.fh.ooe.swe4.fx.campina.jpa.Menu;
import at.fh.ooe.swe4.fx.campina.jpa.MenuEntry;
import at.fh.ooe.swe4.fx.campina.view.annotation.FormField;
import at.fh.ooe.swe4.fx.campina.view.annotation.SelectFormField;
import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;

/**
 * The view model which backs the {@link MenuEntry} entity
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class MenuEntryModel extends AbstractViewModel<Integer, MenuEntry> {

  public ObservableList<Menu>  menus;

  /**
   * The converter for the {@link Menu} type
   *
   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
   * @date Jun 5, 2015
   */
  public static class MenuConverter extends StringConverter<Menu> {

    @Override
    public String toString(Menu object) {
      return (object.getId() == null) ? "Bitte Wählen" : object.getLabel();
    }

    @Override
    public Menu fromString(String string) {
      throw new UnsupportedOperationException("From text not supported by this converter");
    }

  }

  /**
   * The converter for the {@link MenuEntryModel} type.
   *
   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
   * @date Jun 5, 2015
   */
  public static class MenuEntryModelConverter extends StringConverter<MenuEntryModel> {

    @Override
    public String toString(MenuEntryModel object) {
      Objects.requireNonNull(object);
      return (object.getId() == null) ? "Neuer Menu Eintrag" : object.getLabel();
```

```
58        }
59
60      @Override
61      public MenuEntryModel fromString(String string) {
62        throw new UnsupportedOperationException("Form text not supported by this converter");
63      }
64
65    }
66
67    /**
68     *
69     */
70    public MenuEntryModel() {
71      super();
72    }
73
74    /**
75     * @param entity
76     */
77    public MenuEntryModel(MenuEntry entity) {
78      super(entity);
79    }
80
81    @Override
82    public void reset() {
83      final MenuEntry entry = new MenuEntry();
84      entry.setPrice(BigDecimal.ZERO);
85      entry.setMenu(new Menu());
86      entry.setOrdinal(0);
87      menus = FXCollections.observableArrayList();
88      menus.add(new Menu());
89      prepare(entry);
90
91    }
92
93    @FormField(id = "menu-entry-label",
94        label = "Beschreibung",
95        ordinal = 1,
96        required = true,
97        requiredMessage = "Bitte Beschreibung angeben",
98        type = FormFieldType.INPUT_TEXT)
99    public String getLabel() {
100     return getEntity().getLabel();
101   }
102
103   public void setLabel(String label) {
104     getEntity().setLabel(label);
105   }
106
107   @FormField(id = "menu-entry-price",
108       label = "Preis",
109       ordinal = 2,
110       required = true,
111       requiredMessage = "Bitte Preis angeben",
112       type = FormFieldType.DECIMAL)
113   public BigDecimal getPrice() {
114     return getEntity().getPrice();
115   }
116
117   public void setPrice(BigDecimal price) {
118     getEntity().setPrice(price);
119   }
120
```

```java
    @FormField(id = "menu-entry-menu",
        label = "Menu",
        ordinal = 3,
        required = true,
        requiredMessage = "Bitte Menu wählen",
        type = FormFieldType.SELECT,
        valueClass = Menu.class)
    public Menu getMenu() {
      return getEntity().getMenu();
    }

    public void setMenu(Menu menu) {
      getEntity().setMenu(menu);
      if (menu != null) {
        getEntity().setOrdinal(menu.getEntries()
                    .size() + 1);
      }
    }

    @SelectFormField(target = "menu",
        converter = MenuConverter.class)
    public ObservableList<Menu> getMenus() {
      return menus;
    }
}
```

1.9.4 MenuEntryEventControl.java

Folgend ist die Klasse angeführt, welche die Actions behandelt, wie Speichern, Löschen, usw.

Listing 32: MenuEntryEventControl.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.menu.control;

import java.util.Objects;

import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.scene.Node;
import javafx.scene.control.ChoiceBox;
import javafx.scene.text.Text;
import javafx.scene.text.TextFlow;

import org.apache.commons.lang.StringUtils;

import at.fh.ooe.swe4.fx.campina.jpa.EntityCache;
import at.fh.ooe.swe4.fx.campina.jpa.Menu;
import at.fh.ooe.swe4.fx.campina.jpa.MenuEntry;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuEntryModel;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuModel;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.part.MenuTabViewHandler;
import at.fh.ooe.swe4.fx.campina.view.api.FormContext;

/**
 * The event handler for the {@link MenuEntry} entity.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class MenuEntryEventControl {

  public MenuEntryEventControl() {
    // TODO Auto-generated constructor stub
  }

  /**
   * Handles the new action of the form.
   *
   * @param event
   *            the {@link ActionEvent}
   */
  public void newAction(final ActionEvent event) {
    final FormContext<MenuEntryModel> ctx = (FormContext<MenuEntryModel>) ((Node)
      event.getSource()).getUserData();
    // clear former set message
    populateFormMessage(null, ctx);
    // reset form
    ctx.formHandler.resetForm(ctx);
    // create new user model with new user entity
    ctx.model.reset();
    // hide buttons
    ctx.getNode(MenuTabViewHandler.MENU_ENTRY_DELETE_BUTTON_ID)
      .setVisible(Boolean.FALSE);
    // reload users
    handleMenuEntryReload(ctx);
  }

  /**
   * Saves a {@link MenuEntry}
   *
```

```
58      * @param event
59      *           the {@link ActionEvent}
60      */
61    public void saveMenuEntry(final ActionEvent event) {
62      final FormContext<MenuEntryModel> ctx = (FormContext<MenuEntryModel>) ((Node)
        ↪  event.getSource()).getUserData();
63      ctx.getNode(MenuTabViewHandler.MENU_ENTRY_DELETE_BUTTON_ID)
64        .setVisible(Boolean.TRUE);
65      ctx.formHandler.validateForm(ctx);
66      ctx.formHandler.fillModel(ctx);
67
68      if (ctx.valid) {
69        final MenuEntry entry;
70        if (ctx.model.getId() != null) {
71          entry = ctx.model.getEntity();
72          EntityCache.menuEntryCache.add(entry);
73          EntityCache.byMenuId(entry.getMenu()
74                        .getId())
75              .getEntries()
76              .add(entry);
77        } else {
78          entry = ctx.model.getEntity();
79          entry.setId(EntityCache.menuCache.size() + 1);
80          EntityCache.menuEntryCache.add(entry);
81        }
82        ctx.model.prepare(entry);
83        handleMenuEntryReload(ctx);
84      } else {
85        populateFormMessage("Formular ungültig !! Bitte Eingaben prüfen", ctx);
86        handleMenuEntryReload(ctx);
87      }
88    }
89
90    /**
91     * Deletes a {@link MenuEntry}
92     *
93     * @param event
94     *           the {@link ActionEvent}
95     */
96    public void deleteMenuEntry(final ActionEvent event) {
97      final FormContext<MenuEntryModel> ctx = (FormContext<MenuEntryModel>) ((Node)
        ↪  event.getSource()).getUserData();
98
99      if (ctx.model.getId() != null) {
100       ctx.getNode(MenuTabViewHandler.MENU_ENTRY_DELETE_BUTTON_ID)
101         .setVisible(Boolean.TRUE);
102       final MenuEntry entry = ctx.model.getEntity();
103       EntityCache.menuEntryCache.remove(entry);
104       EntityCache.byMenuId(ctx.model.getEntity()
105                     .getMenu()
106                     .getId())
107           .getEntries()
108           .remove(entry);
109       ctx.formHandler.resetForm(ctx);
110       ctx.model.reset();
111     }
112     handleMenuEntryReload(ctx);
113   }
114
115   /**
116    * Handles the {@link MenuEntry} reload
117    */
118   public void handleMenuEntryReload(final FormContext<MenuEntryModel> ctx) {
```

```
119      Objects.requireNonNull(ctx);
120
121      final ObservableList<MenuEntryModel> list = (ObservableList<MenuEntryModel>)
          ↪   ctx.getObserable(MenuTabViewHandler.MENU_ENTRY_SELECTION_KEY);
122      list.clear();
123      list.add(new MenuEntryModel());
124      for (MenuEntry entry : EntityCache.menuEntryCache) {
125        final MenuEntryModel model = new MenuEntryModel();
126        model.prepare(entry);
127        list.add(model);
128      }
129
130      // need to replace observed instance
131      list.set(list.indexOf(ctx.model), ctx.model);
132
133      // need to select current context hold model
134      ((ChoiceBox<MenuEntryModel>)
          ↪   ctx.getNode(MenuTabViewHandler.MENU_ENTRY_SELECTION_KEY)).getSelectionModel()
135                                         .select(ctx.model);
136    }
137
138    /**
139     * Handles the {@link Menu} reload.
140     *
141     * @param ctx
142     *            teh form context
143     */
144    public void handleMenuLoad(final FormContext<MenuEntryModel> ctx) {
145      Objects.requireNonNull(ctx);
146
147      ctx.model.getMenus()
148            .clear();
149      ctx.model.getMenus()
150            .add(new Menu());
151      for (Menu menu : EntityCache.menuCache) {
152        ctx.model.getMenus()
153              .add(menu);
154      }
155
156      ctx.model.getMenus()
157            .set(ctx.model.getMenus()
158                  .indexOf(ctx.model.getMenu()), ctx.model.getMenu());
159    }
160
161    /**
162     * Populates a message to the message box. <br>
163     * If message is null the actual set message will be cleared
164     *
165     * @param message
166     *            the message to populate
167     * @param ctx
168     *            the form context
169     */
170    private void populateFormMessage(final String message, final FormContext<MenuEntryModel> ctx) {
171      final TextFlow flow = ((TextFlow) ctx.getNode(MenuTabViewHandler.MENU_ENTRY_FORM_MESSAGE));
172      flow.getChildren()
173        .clear();
174      flow.setPrefHeight(0);
175      if (!StringUtils.isEmpty(message)) {
176        flow.getChildren()
177          .add(new Text(message));
178        flow.setPrefHeight(30);
179      }
```

```
180        }
181    }
```

1.9.5 MenuTabViewHandler.java

Folgend ist der ViewHanddler der MenuTab angeführt. Sie stellt einen View Teil der Scene dar und implementiert `ViewHandler<T>` Interface.

Listing 33: MenuTabViewHandler.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.menu.part;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;

import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.RadioButton;
import javafx.scene.control.Tab;
import javafx.scene.control.Toggle;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.GridPane;
import javafx.scene.text.TextFlow;
import javafx.util.StringConverter;
import at.fh.ooe.swe4.fx.campina.jpa.Menu;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.control.MenuEntryEventControl;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.control.MenuEventControl;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuEntryModel;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuEntryModel.MenuEntryModelConverter;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuModel;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuModel.MenuModelConverter;
import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
import at.fh.ooe.swe4.fx.campina.view.api.ViewHandler;
import at.fh.ooe.swe4.fx.campina.view.form.FormHandler;

/**
 * The view handler for the menu and menu entry view.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class MenuTabViewHandler implements ViewHandler<Tab> {

  /**
   * The form handler for the menu entry form
   */
  private final FormHandler<MenuEntryModel>  menuEntryFormHandler;
  /**
   * The menu entry controller
   */
  private final MenuEntryEventControl     menuEntryControl;
  /**
   * The form context for the menu entry form
   */
  final FormContext<MenuEntryModel>      menuEntryCtx;

  /**
   * The form handler for the menu form
   */
```

```
57    private final FormHandler<MenuModel>     menuFormHandler;
58    /**
59     * The menu controller
60     */
61    private final MenuEventControl        menuControl;
62    /**
63     * The form context for the menu entry form
64     */
65    final FormContext<MenuModel>        menuCtx;
66
67    // ##########################################################################
68    // Ids of nodes and form backed model class
69    // ##########################################################################
70    public static final String           MENU_ENTRY_FORM_MESSAGE    = "tab-menu-entry-form-message";
71    public static final String           MENU_FORM_MESSAGE      = "tab-menu-form-message";
72    public static final String           EDIT_OPTION_ID        = "tab-menu-edit-option";
73    public static final String           MENU_SELECTION_KEY      = "tab-menu-menu-selection-data";
74    public static final String           MENU_ENTRY_SELECTION_KEY  =
    ↪    "tab-menu-menu-entry-selection-data";
75    public static final String           MENU_SAVE_BUTTON_ID      = "tab-menu-menu-form-save-button";
76    public static final String           MENU_DELETE_BUTTON_ID    =
    ↪    "tab-menu-menu-form-delete-button";
77    public static final String           MENU_ENTRY_SAVE_BUTTON_ID  =
    ↪    "tab-menu-menu-entry-form-save-button";
78    public static final String           MENU_ENTRY_DELETE_BUTTON_ID  =
    ↪    "tab-menu-menu-entry-form-delete-button";
79    public static final String           MENU_NEW_BUTTON_ID      = "tab-menu-menu-form-new-button";
80    public static final String           MENU_ENTRY_NEW_BUTTON_ID  =
    ↪    "tab-menu-menu-entry-form-new-button";
81
82    /**
83     * Enumeration which specifies the edit type. Provides an label too.
84     *
85     * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
86     * @date Jun 5, 2015
87     */
88    public static enum EditMode {
89      MENU("Menus"),
90      MENU_ENTRY("Menu Einträge");
91
92      public final String  label;
93
94      /**
95       * @param label
96       */
97      private EditMode(final String label) {
98        this.label = label;
99      }
100   }
101
102   /**
103    * The converter for the {@link EditMode} type.
104    *
105    * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
106    * @date Jun 5, 2015
107    */
108   public static class EditModeConverter extends StringConverter<EditMode> {
109
110     @Override
111     public String toString(EditMode object) {
112       return object.label;
113     }
114
```

```
115       @Override
116       public EditMode fromString(String string) {
117         throw new UnsupportedOperationException("From string not supportedby this converter");
118       }
119
120   }
121
122   /**
123    * The change listener for the {@link Toggle} type.
124    *
125    * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
126    * @date Jun 5, 2015
127    */
128   public static class ChangeModeListener implements ChangeListener<Toggle> {
129
130     private final GridPane          parent;
131     private final GridPane          menuGroup;
132     private final GridPane          menuEntryGroup;
133     private final MenuEventControl     menuControl;
134     private final MenuEntryEventControl  menuEntryControl;
135
136     /**
137      * @param parent
138      * @param menuGroup
139      * @param menuEntryGroup
140      * @param menuControl
141      * @param menuEntryControl
142      */
143     public ChangeModeListener(GridPane parent, GridPane menuGroup, GridPane menuEntryGroup,
         ↪   MenuEventControl menuControl,
144         MenuEntryEventControl menuEntryControl) {
145       super();
146       Objects.requireNonNull(parent);
147       Objects.requireNonNull(menuGroup);
148       Objects.requireNonNull(menuEntryGroup);
149       Objects.requireNonNull(menuControl);
150       Objects.requireNonNull(menuEntryControl);
151
152       this.parent = parent;
153       this.menuGroup = menuGroup;
154       this.menuEntryGroup = menuEntryGroup;
155       this.menuControl = menuControl;
156       this.menuEntryControl = menuEntryControl;
157     }
158
159     @Override
160     public void changed(ObservableValue<? extends Toggle> observable, Toggle oldValue, Toggle
         ↪   newValue) {
161       if (observable.getValue() == null) {
162         return;
163       }
164
165       parent.getChildren()
166           .remove(menuEntryGroup);
167       parent.getChildren()
168           .remove(menuGroup);
169
170       switch ((EditMode) observable.getValue()
171                     .getUserData()) {
172       case MENU:
173         parent.add(menuGroup, 0, 1);
174         final FormContext<MenuModel> menuCtx = (FormContext<MenuModel>) menuGroup.getUserData();
175         menuCtx.model.reset();
```

```
176         menuControl.handleMenuReload(menuCtx);
177         ((ChoiceBox<MenuModel>) menuCtx.getNode(MENU_SELECTION_KEY)).getSelectionModel()
178                               .select(menuCtx.model);
179         menuCtx.getNode(MENU_DELETE_BUTTON_ID)
180             .setVisible(Boolean.FALSE);
181         menuCtx.formHandler.fillForm(menuCtx);
182         break;
183       case MENU_ENTRY:
184         parent.add(menuEntryGroup, 0, 1);
185         final FormContext<MenuEntryModel> menuEntryCtx = (FormContext<MenuEntryModel>)
            ↪   menuEntryGroup.getUserData();
186         menuEntryCtx.model.reset();
187         menuEntryControl.handleMenuEntryReload(menuEntryCtx);
188         ((ChoiceBox<MenuEntryModel>)
            ↪   menuEntryCtx.getNode(MENU_ENTRY_SELECTION_KEY)).getSelectionModel()
189                                 .select(menuEntryCtx.model);
190         menuEntryCtx.getNode(MENU_ENTRY_DELETE_BUTTON_ID)
191             .setVisible(Boolean.FALSE);
192         menuEntryControl.handleMenuLoad(menuEntryCtx);
193         menuEntryCtx.formHandler.fillForm(menuEntryCtx);
194         break;
195       default:
196         throw new IllegalArgumentException("'EditMode#" + ((EditMode) observable.getValue()
197                                 .getUserData()).name() + "' node suppported");
198     }
199   }
200 }
201
202 /**
203  * The change listener for the {@link MenuModel} type.
204  *
205  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
206  * @date Jun 5, 2015
207  */
208 public static class MenuChangeListener implements ChangeListener<MenuModel> {
209
210   private final FormContext<MenuModel>  ctx;
211   private final MenuEventControl      control;
212
213   /**
214    * @param ctx
215    * @param control
216    */
217   public MenuChangeListener(FormContext<MenuModel> ctx, MenuEventControl control) {
218     super();
219     Objects.requireNonNull(ctx);
220     Objects.requireNonNull(control);
221
222     this.ctx = ctx;
223     this.control = control;
224   }
225
226   @Override
227   public void changed(ObservableValue<? extends MenuModel> observable, MenuModel oldValue,
        ↪   MenuModel newValue) {
228     if (observable.getValue() == null) {
229       return;
230     }
231
232     ctx.model = observable.getValue();
233     ctx.model.prepare(observable.getValue()
234                   .getEntity());
235     if (ctx.model.getId() != null) {
```

```
236            ctx.getNode(MENU_DELETE_BUTTON_ID)
237                .setVisible(Boolean.TRUE);
238          } else {
239            ctx.getNode(MENU_DELETE_BUTTON_ID)
240                .setVisible(Boolean.FALSE);
241          }
242          ctx.formHandler.fillForm(ctx);
243      }
244    }
245
246    /**
247     * The change listener for the {@link MenuEntryModel} type
248     *
249     * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
250     * @date Jun 5, 2015
251     */
252    public static class MenuEntryChangeListener implements ChangeListener<MenuEntryModel> {
253
254      private final FormContext<MenuEntryModel>  ctx;
255      private final MenuEntryEventControl        control;
256
257      /**
258       * @param ctx
259       * @param control
260       */
261      public MenuEntryChangeListener(FormContext<MenuEntryModel> ctx, MenuEntryEventControl control)
         ↪   {
262        super();
263        Objects.requireNonNull(ctx);
264        Objects.requireNonNull(control);
265
266        this.ctx = ctx;
267        this.control = control;
268      }
269
270      @Override
271      public void changed(ObservableValue<? extends MenuEntryModel> observable, MenuEntryModel
         ↪   oldValue, MenuEntryModel newValue) {
272        if (observable.getValue() == null) {
273          return;
274        }
275
276        ctx.model = observable.getValue();
277        ctx.model.prepare(observable.getValue()
278                    .getEntity());
279        control.handleMenuLoad(ctx);
280        if (ctx.model.getId() == null) {
281          ctx.getNode(MENU_ENTRY_DELETE_BUTTON_ID)
282            .setVisible(Boolean.FALSE);
283        } else {
284          ctx.getNode(MENU_ENTRY_DELETE_BUTTON_ID)
285            .setVisible(Boolean.TRUE);
286        }
287        ctx.formHandler.fillForm(ctx);
288      }
289    }
290
291    /**
292     * @param scene
293     */
294    public MenuTabViewHandler(final Scene scene) {
295      Objects.requireNonNull(scene);
296
```

```
297    this.menuEntryControl = new MenuEntryEventControl();
298    this.menuEntryFormHandler = new FormHandler<MenuEntryModel>();
299    this.menuEntryFormHandler.init();
300    this.menuEntryCtx = new FormContext<MenuEntryModel>("menu-entry-form", menuEntryFormHandler,
       ↪   new MenuEntryModel(), scene);
301
302    this.menuControl = new MenuEventControl();
303    this.menuFormHandler = new FormHandler<MenuModel>();
304    this.menuFormHandler.init();
305    this.menuCtx = new FormContext<MenuModel>("menu-form", menuFormHandler, new MenuModel(),
       ↪   scene);
306    }
307
308    @Override
309    public String getId() {
310      return "tab-menu";
311    }
312
313    @Override
314    public Tab createNode() {
315      // edit mode toggle
316      final ToggleGroup group = new ToggleGroup();
317      final List<RadioButton> buttons = createEditModeToggle(group);
318      final GridPane editOptionsGrid = new GridPane();
319      editOptionsGrid.setId(getId() + "-edit-options-grid");
320      editOptionsGrid.setHgap(10);
321      editOptionsGrid.setVgap(10);
322      for (int i = 0; i < buttons.size(); i++) {
323        editOptionsGrid.add(buttons.get(i), i, 0);
324      }
325      group.getToggles()
326          .get(0)
327          .setSelected(Boolean.TRUE);
328
329      // Form message text
330      final TextFlow menuEntryFlow = new TextFlow();
331      menuEntryFlow.setId(MENU_ENTRY_FORM_MESSAGE);
332      menuEntryFlow.setStyle("-fx-font-size: 20pt");
333      menuEntryCtx.putNode(MENU_ENTRY_FORM_MESSAGE, menuEntryFlow);
334
335      final TextFlow menuFlow = new TextFlow();
336      menuEntryFlow.setId(MENU_ENTRY_FORM_MESSAGE);
337      menuEntryFlow.setStyle("-fx-font-size: 20pt");
338      menuCtx.putNode(MENU_FORM_MESSAGE, menuFlow);
339
340      // menu form grid
341      final GridPane menuGrid = new GridPane();
342      menuGrid.setId(getId() + "-menu-grid");
343      menuGrid.add(menuFlow, 0, 0);
344      menuGrid.add(createMenuChoiceBox(menuCtx), 0, 1);
345      menuGrid.add(menuCtx.formHandler.generateFormGrid(menuCtx), 0, 2);
346      menuGrid.add(createMenuButtonGroup(menuCtx), 0, 3);
347      menuGrid.setUserData(menuCtx);
348      menuGrid.setHgap(10);
349      menuGrid.setVgap(10);
350
351      final GridPane menuEntryGrid = new GridPane();
352      menuEntryGrid.setId(getId() + "-menu-entry-grid");
353      menuEntryGrid.add(menuEntryFlow, 0, 0);
354      menuEntryGrid.add(createMenuEntryChoiceBox(menuEntryCtx), 0, 1);
355      menuEntryGrid.add(menuEntryCtx.formHandler.generateFormGrid(menuEntryCtx), 0, 2);
356      menuEntryGrid.add(createMenuEntryButtonGroup(menuEntryCtx), 0, 3);
357      menuEntryGrid.setUserData(menuEntryCtx);
```

```java
358        menuEntryGrid.setHgap(10);
359        menuEntryGrid.setVgap(10);
360
361        // content grid
362        final GridPane gridPane = new GridPane();
363        gridPane.setId(getId() + "-content");
364        gridPane.add(editOptionsGrid, 0, 0);
365        gridPane.add(menuGrid, 0, 1);
366        gridPane.setHgap(10);
367        gridPane.setVgap(10);
368
369        // TODO: Set change listener on edit mode choice;
370        group.selectedToggleProperty()
371            .addListener(new ChangeModeListener(gridPane, menuGrid, menuEntryGrid, menuControl,
                ↪   menuEntryControl));
372
373        final Tab menuTab = new Tab();
374        menuTab.setId("tab-menu");
375        menuTab.setText("Menus");
376        menuTab.setClosable(Boolean.FALSE);
377
378        menuTab.setContent(gridPane);
379
380        return menuTab;
381    }
382
383    /**
384     * Creates the edit toogle for the edit mode.
385     *
386     * @param group
387     *            the toogle group which holds the toggles.
388     * @return the list of toggles
389     */
390    private List<RadioButton> createEditModeToggle(final ToggleGroup group) {
391        Objects.requireNonNull(group);
392
393        final List<RadioButton> buttons = new ArrayList<>();
394
395        for (EditMode editMode : EditMode.values()) {
396            final RadioButton button = new RadioButton();
397            button.setId(getId() + "-edit-mode-" + editMode.name()
398                            .toLowerCase());
399            button.setText(editMode.label);
400            button.setToggleGroup(group);
401            button.setUserData(editMode);
402            buttons.add(button);
403        }
404
405        return buttons;
406    }
407
408    /**
409     * Creates a menu choice box.
410     *
411     * @param ctx
412     *            the form context
413     * @return the choice box
414     */
415    private ChoiceBox<MenuModel> createMenuChoiceBox(final FormContext<MenuModel> ctx) {
416        Objects.requireNonNull(ctx);
417
418        final ObservableList<MenuModel> menus = FXCollections.observableArrayList(Arrays.asList(new
           ↪   MenuModel()));
```

```
419
420      final ChoiceBox<MenuModel> menuChoice = new ChoiceBox<>(menus);
421      menuChoice.setUserData(ctx);
422      menuChoice.getSelectionModel()
423            .selectedItemProperty()
424            .addListener(new MenuChangeListener(ctx, menuControl));
425      menuChoice.setConverter(new MenuModelConverter());
426      menuChoice.setPrefWidth(400);
427
428      ctx.putObserable(MENU_SELECTION_KEY, menus);
429      ctx.putNode(MENU_SELECTION_KEY, menuChoice);
430
431      return menuChoice;
432    }
433
434    /**
435     * Creates a menu entry choice box
436     *
437     * @param ctxthe
438     *            form context
439     * @return the choice box
440     */
441    private ChoiceBox<MenuEntryModel> createMenuEntryChoiceBox(final FormContext<MenuEntryModel>
      ↪   ctx) {
442      Objects.requireNonNull(ctx);
443
444      final ObservableList<MenuEntryModel> entries =
      ↪   FXCollections.observableArrayList(Arrays.asList(new MenuEntryModel()));
445
446      final ChoiceBox<MenuEntryModel> menuEntryChoice = new ChoiceBox<>(entries);
447      menuEntryChoice.setUserData(ctx);
448      menuEntryChoice.getSelectionModel()
449            .selectedItemProperty()
450            .addListener(new MenuEntryChangeListener(ctx, menuEntryControl));
451      menuEntryChoice.setConverter(new MenuEntryModelConverter());
452      menuEntryChoice.setPrefWidth(400);
453
454      ctx.putObserable(MENU_ENTRY_SELECTION_KEY, entries);
455      ctx.putNode(MENU_ENTRY_SELECTION_KEY, menuEntryChoice);
456
457      return menuEntryChoice;
458    }
459
460    /**
461     * Creates the menu button group
462     *
463     * @param ctx
464     *            the form context
465     * @return the creates button group
466     */
467    private GridPane createMenuButtonGroup(final FormContext<MenuModel> ctx) {
468      Objects.requireNonNull(ctx);
469
470      final GridPane gridPane = new GridPane();
471      gridPane.setId(getId() + "-button-grid");
472      gridPane.setHgap(10);
473      gridPane.setVgap(10);
474
475      final Button newButton = new Button();
476      newButton.setText("Neues Menu");
477      newButton.setId(MENU_NEW_BUTTON_ID);
478      newButton.setUserData(ctx);
479      newButton.setOnAction(menuControl::newAction);
```

```
480
481        final Button saveButton = new Button();
482        saveButton.setText("Speichern");
483        saveButton.setId(MENU_SAVE_BUTTON_ID);
484        saveButton.setUserData(ctx);
485        saveButton.setOnAction(menuControl::saveMenu);
486
487        final Button deleteButton = new Button();
488        deleteButton.setId(MENU_DELETE_BUTTON_ID);
489        deleteButton.setText("Löschen");
490        deleteButton.setUserData(ctx);
491        deleteButton.setOnAction(menuControl::deleteMenu);
492
493        gridPane.add(newButton, 0, 0);
494        gridPane.add(saveButton, 1, 0);
495        gridPane.add(deleteButton, 2, 0);
496
497        // register in context
498        ctx.putNode(MENU_NEW_BUTTON_ID, newButton);
499        ctx.putNode(MENU_SAVE_BUTTON_ID, saveButton);
500        ctx.putNode(MENU_DELETE_BUTTON_ID, deleteButton);
501
502        return gridPane;
503    }
504
505    /**
506     * Creates the menu entry button group
507     *
508     * @param ctx
509     *            the from context
510     * @return the created button group
511     */
512    private GridPane createMenuEntryButtonGroup(final FormContext<MenuEntryModel> ctx) {
513        Objects.requireNonNull(ctx);
514
515        final GridPane gridPane = new GridPane();
516        gridPane.setId(getId() + "-button-grid");
517        gridPane.setHgap(10);
518        gridPane.setVgap(10);
519
520        final Button newButton = new Button();
521        newButton.setText("Neuer Menu Eintrag");
522        newButton.setId(MENU_ENTRY_NEW_BUTTON_ID);
523        newButton.setUserData(ctx);
524        newButton.setOnAction(menuEntryControl::newAction);
525
526        final Button saveButton = new Button();
527        saveButton.setText("Speichern");
528        saveButton.setId(MENU_ENTRY_SAVE_BUTTON_ID);
529        saveButton.setUserData(ctx);
530        saveButton.setOnAction(menuEntryControl::saveMenuEntry);
531
532        final Button deleteButton = new Button();
533        deleteButton.setId(MENU_ENTRY_DELETE_BUTTON_ID);
534        deleteButton.setText("Löschen");
535        deleteButton.setUserData(ctx);
536        deleteButton.setOnAction(menuEntryControl::deleteMenuEntry);
537
538        gridPane.add(newButton, 0, 0);
539        gridPane.add(saveButton, 1, 0);
540        gridPane.add(deleteButton, 2, 0);
541
542        // register in context
```

```
543        ctx.putNode(MENU_ENTRY_NEW_BUTTON_ID, newButton);
544        ctx.putNode(MENU_ENTRY_SAVE_BUTTON_ID, saveButton);
545        ctx.putNode(MENU_ENTRY_DELETE_BUTTON_ID, deleteButton);
546
547        return gridPane;
548    }
549
550    @Override
551    public void initHandler() {
552        menuControl.handleMenuReload(menuCtx);
553        menuCtx.formHandler.fillForm(menuCtx);
554        ((ChoiceBox<MenuModel>) menuCtx.getNode(MENU_SELECTION_KEY)).getSelectionModel()
555                                        .select(menuCtx.model);
556        menuCtx.getNode(MENU_DELETE_BUTTON_ID)
557            .setVisible(Boolean.FALSE);
558    }
559
560 }
```

1.10 Source-Code(Implementation Order View)

Folgend ist der Source der Order View angeführt.

1.10.1 OrderModel.java

Folgende Klasse representiert das View Model für Order Entitäten.

Listing 34: OrderModel.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.order.model;

import org.apache.commons.lang.time.DateFormatUtils;

import at.fh.ooe.swe4.fx.campina.jpa.Order;
import at.fh.ooe.swe4.fx.campina.jpa.User;
import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;

/**
 * The view model backing the {@link Order} entity.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class OrderModel extends AbstractViewModel<Integer, Order> {

  /**
   *
   */
  public OrderModel() {
    super();
  }

  /**
   * @param entity
   */
  public OrderModel(Order entity) {
    super(entity);
  }

  @Override
  public void reset() {
    prepare(new Order());
  }

  // Resolved by TableColumn
  public String getOrderDate() {
    return DateFormatUtils.ISO_DATETIME_FORMAT.format(getEntity().getOrderDate());
  }

  public String getUsername() {
    final User user = getEntity().getUser();
    return new StringBuilder(user.getLastName()).append(", ")
                         .append(user.getFirstName())
                         .toString();
  }

  public String getMenuLabel() {
    return getEntity().getMenuEntry()
               .getMenu()
               .getLabel();
  }

```

```
54    public String getMenuEntryLabel() {
55      return getEntity().getMenuEntry()
56              .getLabel();
57    }
58  }
```

1.10.2   OrderEventHandler.java

Folgend ist die Klasse angeführt, welche die Actions behandelt, wie Speichern, Löschen, usw.

Listing 35: OrderEventHandler.java

```java
package at.fh.ooe.swe4.fx.campina.view.admin.order.control;

import java.util.Objects;

import javafx.collections.ObservableList;
import at.fh.ooe.swe4.fx.campina.jpa.EntityCache;
import at.fh.ooe.swe4.fx.campina.jpa.Order;
import at.fh.ooe.swe4.fx.campina.view.admin.order.model.OrderModel;

public class OrderEventHandler {

  public OrderEventHandler() {
    // TODO Auto-generated constructor stub
  }

  public void reloadOrders(ObservableList<OrderModel> list) {
    Objects.requireNonNull(list);

    list.clear();
    for (Order order : EntityCache.orderCache) {
      list.add(new OrderModel(order));
    }
  }
}
```

1.11 Source-Code(Implementation Main Scene)

Folgend ist der Source der Main Scene angeführt.

1.11.1 MainSceneViewHandler.java

Diese Klasse baut sich die Scene auf initialisiert sie. In Zukunft könnte sie auch das wechseln zwischen View Teilen verwalten (Tab Visibility, usw.).
Dies könnte über Menu-Einträge erfolgen, die hier erstellt und verwaltet werden.

Listing 36: MainSceneViewHandler.java

```java
package at.fh.ooe.swe4.fx.campina.view.scene;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;

import javafx.scene.Scene;
import javafx.scene.control.Tab;
import javafx.scene.control.TabPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import at.fh.ooe.swe4.fx.campina.component.builder.impl.MenuBarBuilder;
import at.fh.ooe.swe4.fx.campina.component.builder.impl.MenuBuilder;
import at.fh.ooe.swe4.fx.campina.component.builder.impl.MenuItemBuilder;
import at.fh.ooe.swe4.fx.campina.view.admin.login.part.LoginTabViewHandler;
import at.fh.ooe.swe4.fx.campina.view.admin.menu.part.MenuTabViewHandler;
import at.fh.ooe.swe4.fx.campina.view.admin.order.part.OrderTabViewHandler;
import at.fh.ooe.swe4.fx.campina.view.admin.user.part.UserTabviewHandler;
import at.fh.ooe.swe4.fx.campina.view.api.EventHandlerFactory;
import at.fh.ooe.swe4.fx.campina.view.api.ViewHandler;

/**
 * The view ahdnler for the main scene.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 4, 2015
 */
public class MainSceneViewHandler implements ViewHandler<Scene> {

  private static final long  serialVersionUID  = -7630460091326559133L;

  private UserTabviewHandler  userTab;
  private LoginTabViewHandler  loginTab;
  private MenuTabViewHandler  menuTab;
  private OrderTabViewHandler  orderTab;

  /**
   * Enumeration which specifies the menus placed in the menu bar.
   *
   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
   * @date May 29, 2015
   */
  public static enum MenuDefinition {
    FILE(1, "menu-file", "Dateien");

    public final int           ordinal;
    public final String           id;
    public final String           label;
```

```
51      public static final EventHandlerFactory  eventHandlerFactory  = null;
52
53      /**
54       * @param ordinal
55       * @param id
56       * @param label
57       */
58      private MenuDefinition(int ordinal, String id, String label) {
59        this.ordinal = ordinal;
60        this.id = id;
61        this.label = label;
62      }
63    }
64
65    /**
66     * This enumeration specifies the menu items for the specified menus.
67     *
68     * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
69     * @date May 29, 2015
70     */
71    public static enum MenuItemDefinition {
72      /*-- User menu items --*/
73      CLOSE(3, "item-close", "Schließen", MenuDefinition.FILE);
74
75      public final String              id;
76      public final String              label;
77      public final Integer             ordinal;
78      public final MenuDefinition          menu;
79      public static final EventHandlerFactory<String>  eventHandlerFactory  = new
         ↪   MainSceneEventHandlerFactory();
80
81      /**
82       * @param ordinal
83       * @param id
84       * @param label
85       * @param menu
86       */
87      private MenuItemDefinition(int ordinal, String id, String label, MenuDefinition menu) {
88        this.id = id;
89        this.label = label;
90        this.ordinal = ordinal;
91        this.menu = menu;
92      }
93
94      /**
95       * Filters the provided enumerations for the given menu.
96       *
97       * @param menu
98       *           the menu to get its specified items
99       * @return the menu related items, an empty list otherwise
100      */
101     public static final List<MenuItemDefinition> filterForMenu(final MenuDefinition menu) {
102       final List<MenuItemDefinition> items;
103       if (menu != null) {
104         items = Arrays.asList(MenuItemDefinition.values())
105                 .stream()
106                 .filter((el) -> menu.equals(el.menu))
107                 .collect(Collectors.toList());
108         Collections.sort(items, (o1, o2) -> o1.ordinal.compareTo(o2.ordinal));
109       } else {
110         items = Collections.EMPTY_LIST;
111       }
112       return items;
```

```
113          }
114     }
115
116     /**
117      * Creates a main scene definition instance.<br>
118      * The main components ids are set here.
119      */
120     public MainSceneViewHandler() {
121     }
122
123     @Override
124     public Scene createNode() {
125        final Pane rootPane = new VBox();
126        final Pane menuBarBox = new VBox();
127        final TabPane tabPane = new TabPane();
128
129        rootPane.setId("root-pane");
130        menuBarBox.setId("menu-bar-box");
131        tabPane.setId("tab-pane");
132
133        final Scene scene = new Scene(rootPane);
134        this.userTab = new UserTabviewHandler(scene);
135        this.loginTab = new LoginTabViewHandler(scene);
136        this.menuTab = new MenuTabViewHandler(scene);
137        this.orderTab = new OrderTabViewHandler(scene);
138
139        prepareMenuBox(menuBarBox);
140        prepareTabs(tabPane);
141
142        rootPane.getChildren()
143            .add(menuBarBox);
144        rootPane.getChildren()
145            .add(tabPane);
146
147        return scene;
148     }
149
150     @Override
151     public void initHandler() {
152        // this.loginTab.init();
153        this.userTab.initHandler();
154        this.menuTab.initHandler();
155        this.orderTab.initHandler();
156     }
157
158     /**
159      * Prepares the menu box by adding all menus and its items.
160      *
161      * @param menuBox
162      *            the menu box to provide
163      */
164     public void prepareMenuBox(final Pane menuBox) {
165        Objects.requireNonNull(menuBox, "Cannot prepare null menu box");
166
167        MenuBarBuilder mbb = new MenuBarBuilder();
168        MenuBuilder mb = new MenuBuilder();
169        MenuItemBuilder mib = new MenuItemBuilder();
170
171        mbb.start();
172        for (MenuDefinition menu : MenuDefinition.values()) {
173           mb.start();
174           if (MenuDefinition.eventHandlerFactory != null) {
175              mb.registerEventHandlers(MenuItemDefinition.eventHandlerFactory.registerEventHandler(menu.id));
```

```
176          }
177          for (MenuItemDefinition item : MenuItemDefinition.filterForMenu(menu)) {
178            mib.start();
179            if (MenuItemDefinition.eventHandlerFactory != null) {
180              mib.registerEventHandlers(MenuItemDefinition.eventHandlerFactory.registerEventHandler(item.id));
181            }
182            mb.addItem(mib.build(item.id, item.label));
183            mib.end();
184          }
185          mbb.addMenu(mb.build(menu.id, menu.label));
186          mb.end();
187        }
188        menuBox.getChildren()
189            .add(mbb.build());
190        mbb.end();
191      }
192
193      public void prepareTabs(final TabPane tabPane) {
194        Objects.requireNonNull(tabPane, "Cannot prepare null content box");
195
196        final Tab login = loginTab.createNode();
197        final Tab user = userTab.createNode();
198        final Tab menu = menuTab.createNode();
199        final Tab order = orderTab.createNode();
200        // user.setDisable(Boolean.TRUE);
201        // menu.setDisable(Boolean.TRUE);
202        // order.setDisable(Boolean.TRUE);
203
204        tabPane.getTabs()
205            .add(login);
206        tabPane.getTabs()
207            .add(user);
208        tabPane.getTabs()
209            .add(menu);
210        tabPane.getTabs()
211            .add(order);
212      }
213
214      @Override
215      public String getId() {
216        return "main-scene";
217      }
218    }
```

1.11.2 MainSceneEventHandlerFactory.java

Folgend ist der Source der EVentHandlerFactory für die Main Scene angeführt, diedie Events für die Window Menus erstellt.

Listing 37: MainSceneEventHandlerFactory.java

```java
package at.fh.ooe.swe4.fx.campina.view.scene;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;

import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.event.EventType;
import at.fh.ooe.swe4.fx.campina.view.api.EventHandlerFactory;
import at.fh.ooe.swe4.fx.campina.view.scene.MainSceneViewHandler.MenuItemDefinition;

/**
 * The event handler factory for the main scene.
 *
 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
 * @date Jun 5, 2015
 */
public class MainSceneEventHandlerFactory implements EventHandlerFactory<String> {

  private static final long  serialVersionUID  = 6992457824162264209L;

  /**
   *
   */
  public MainSceneEventHandlerFactory() {
  }

  @SuppressWarnings("rawtypes")
  @Override
  public Map<EventType, List<EventHandler>> registerEventHandler(final String id) {
    Objects.requireNonNull(id, "Cannot register event for null id");

    final Map<EventType, List<EventHandler>> eventHandlers = new HashMap<>();
    eventHandlers.put(ActionEvent.ACTION, new ArrayList<EventHandler>());

    // new user event
    if (id.equals(MenuItemDefinition.CLOSE.id)) {
      eventHandlers.get(ActionEvent.ACTION)
              .add(new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                  System.exit(0);
                }
              });
    }

    return eventHandlers;
  }
}
```