

Gr. 1, E. Pitzer

Name Roman LumetsbergerAufwand in h 7

Gr. 2, F. Gruber-Leitner

Punkte _____ Kurzzeichen Tutor / Übungsleiter _____ / _____

Ausbaustufe 2: CaaS-DB

(24 Punkte)

Nachdem der Restaurantbetreiber von der Präsentation Ihrer ersten Ausbaustufe restlos begeistert war, steht einem Ausbau bis hin zur Online-Plattform nichts mehr im Wege.

Im nächsten Schritt sollen Sie die Anforderung umsetzen, dass mehrere Mitarbeiter des Restaurantbetreibers die Benutzeroberfläche gleichzeitig benutzen können (im Restaurant und auch von zu Hause aus). Dafür ist es notwendig, die Benutzeroberfläche und die Datenverwaltung zu entkoppeln und die Kommunikation der beiden Komponenten über das Netzwerk durchzuführen.

In dieser Ausbaustufe sollen Sie weiters dafür sorgen, dass die Daten dauerhaft in einer Datenbank gespeichert werden.

Im Detail sollte Ihre Anwendung folgende Anforderungen erfüllen:

- Überlegen Sie sich eine geeignete Repräsentation für die Daten in Ihrem Programm. Im konkreten Anwendungsfall werden Sie dafür Klassen zur Speicherung der Benutzer-, Menü- und Bestelldaten benötigen. Diese Klassen werden häufig als Domänenmodell bezeichnet.
- Implementieren Sie einen RMI-Server, der die Daten des Menü-Bestellsystems zentral verwaltet. Stellen Sie sicher, dass die Serverkomponente parallel beliebig viele Clients mit Daten versorgen kann.
- Bauen Sie die in Übung 6 entwickelte JavaFX-basierte Benutzeroberfläche zu einem RMI-Client aus, der mit der Serverkomponente kommuniziert. Entwerfen Sie ein Datenmodell, das alle in CaaS anfallenden Daten abbildet. Berücksichtigen Sie nicht nur die Daten der Menüverwaltung sondern auch die Daten für die Online-Bestellung.
- Stellen Sie Klassen zum Zugriff auf die Datenbank auf Basis von JDBC zur Verfügung. Bereiten Sie auch bereits Methoden für das Hinzufügen und Laden von Bestellungen vor. Achten Sie darauf, dass die Datenbankzugriffsschicht möglichst weitgehend von den anderen Komponenten der Anwendung getrennt ist. Stellen Sie dazu die gesamte Funktionalität der Datenbankzugriffsschicht über Interfaces zur Verfügung.
- Entwickeln Sie eine Testsuite, mit der Sie die Datenzugriffsschicht unabhängig von den anderen Systemkomponenten testen können.

1 CaaS – Campina as a Service

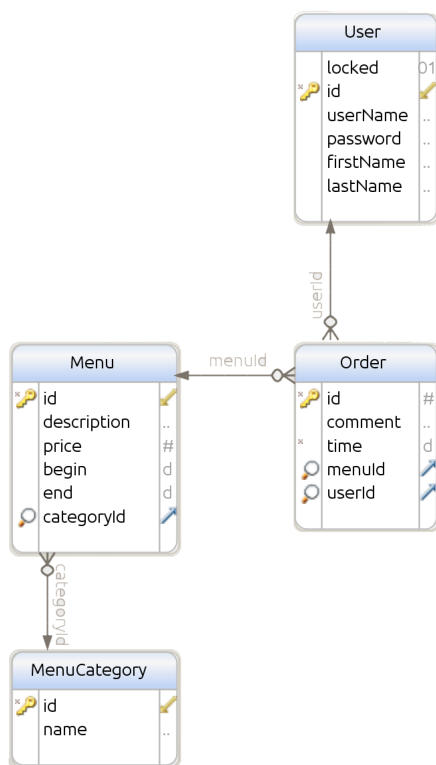
1.1 Lösungsidee

Bei dieser Aufgabe muss das Datenbankschema erstellt und dann pro Tabelle ein *Dao* implementiert werden.

Weiters wird dann der aktuelle BusinessLayer per RMI verwendet. Die neue Implementierung kann die *Dao's* verwenden um die DB Operation durchzuführen.

1.1.1 Datenbank

Folgendes Schema wird für die Datenbank verwendet.



Das Datenbankschema liegt im Projektordner unter **dbscripts/create-tables-mysql.sql**

Achtung: Das Programm benötigt den MYSQL Treiber.

1.1.2 Dao's

Für jede Tabelle des Schemas wird ein Dao Interface erstellt, wobei dies von einem generischen Basisinterface abgeleitet werden kann.

Die JDBC Implementierung kann generisch über *Reflection* umgesetzt werden. Auch hier kann wieder eine generische Basisklasse verwendet werden.

1.1.3 BusinessLayer - Dao Implementierung

Der **Business-Layer** verwendet die **Dao's** um die benötigten Operationen auszuführen.

1.1.4 RMI

Die RMI Implementierung wurde der Einfachheit halber direkt in das Projekt integriert, sollte normalerweise aber in ein eigenes Projekt ausgelagert werden.

Da in Übung 06 bereits ein Business-Layer verwendet wurde, kann dieser nun einfach per RMI Interface bereitgestellt werden.

Der RMI Server Layer kann über die Klasse **at.lumetsnet.caas.rmi.RemoteServiceMain** gestartet werden.

1.2 Sourcecode - SQL

create-tables-mysql.sql

```
1 create schema if not exists CaasDb;
2 use CaasDb;
3 SET foreign_key_checks = 0;
4
5 drop table if exists User;
6 create table User (id int auto_increment primary key,
7                   userName varchar(50), password varchar(50), firstName varchar(50),
8                   lastName varchar(50), locked BOOL);
9 drop table if exists MenuCategory;
10 create table MenuCategory (id int auto_increment primary key,
11                            name varchar(50));
12
13 drop table if exists Menu;
14 create table Menu (id int auto_increment primary key,
15                   description varchar(50),
16                   price int,
17                   begin date,
18                   end date,
19                   categoryId int,
20                   FOREIGN KEY (categoryId) REFERENCES MenuCategory(id)
21                   ON DELETE CASCADE);
22
23 drop table if exists 'Order';
24 create table 'Order' (id int auto_increment primary key,
25                      comment varchar(100),
26                      time timestamp,
27                      menuId int,
28                      userId int,
29                      FOREIGN KEY (menuId) REFERENCES Menu(id),
```

```
30      FOREIGN KEY (userId) REFERENCES User(id)
31      ON DELETE CASCADE);
32
33 SET foreign_key_checks = 1;
```

1.3 Sourcecode - Java

Main.java

```
1 package at.lumetsnet.caas;
2
3 import java.util.logging.LogManager;
4
5 import javafx.application.Application;
6 import javafx.stage.Stage;
7 import at.lumetsnet.caas.business.MenuService;
8 import at.lumetsnet.caas.business.OrderService;
9 import at.lumetsnet.caas.business.UserService;
10 import at.lumetsnet.caas.gui.MainWindow;
11 import at.lumetsnet.caas.model.User;
12
13 public class Main extends Application {
14
15     @Override
16     public void start(Stage stage) throws Exception {
17         User user = new User();
18         user.setFirstName("Roman");
19         user.setLastName("Lumetsberger");
20
21         MainWindow wnd = new MainWindow(stage, user);
22         wnd.show();
23
24     }
25
26     public static void main(String[] args) {
27         LogManager.getLogManager().reset();
28
29         String rmiServer = "localhost:1099";
30         if (args.length > 0) {
31             rmiServer = args[0];
32         }
33
34         initializeServices(rmiServer);
35
36         launch(args);
37
38     }
39
40     private static void initializeServices(String rmiServer) {
41         if (!UserService.getInstance().initialize(rmiServer)) {
42             System.out.println("Error initializing user service");
43             System.exit(-1);
44         }
45         if (!OrderService.getInstance().initialize(rmiServer)) {
```

```
46     System.out.println("Error initializing order service");
47     System.exit(-1);
48 }
49 if (!MenuService.getInstance().initialize(rmiServer)) {
50     System.out.println("Error initializing menu service");
51     System.exit(-1);
52 }
53
54 }
55
56 }
```

business/MenuService.java

```
1 package at.lumetsnet.caas.business;
2
3 import java.rmi.RemoteException;
4 import java.util.Collection;
5
6 import at.lumetsnet.caas.model.Menu;
7 import at.lumetsnet.caas.model.MenuCategory;
8 import at.lumetsnet.caas.rmi.interfaces.RemoteMenuService;
9
10 /**
11  * Mock menu business logic class Used as singleton, should be replaced with
12  * real logic class
13  *
14  * @author romanlum
15  *
16  */
17 public class MenuService extends RmiService<RemoteMenuService> {
18
19     private static MenuService instance = null;
20
21     private MenuService() {
22         super("CaasMenuService");
23     }
24
25     /**
26      * Singleton instance class
27      *
28      * @return
29      */
30     public static MenuService getInstance() {
31         if (instance == null) {
32             instance = new MenuService();
33         }
34         return instance;
35     }
36 }
```

```
36
37  /**
38   * Fetches all categories
39   *
40   * @return
41   */
42  public Collection<MenuCategory> getAllCategories() {
43      try {
44          return service.getAllCategories();
45      } catch (RemoteException e) {
46          e.printStackTrace();
47      }
48      return null;
49  }
50
51  /**
52   * Deletes the category with the given id
53   *
54   * @param id
55   */
56  public void deleteCategory(long id) {
57      try {
58          service.deleteCategory(id);
59      } catch (RemoteException e) {
60          e.printStackTrace();
61      }
62      ;
63  }
64
65  /**
66   * Deletes the menu with the given id
67   *
68   * @param id
69   */
70  public void deleteMenu(long id) {
71      try {
72          service.deleteMenu(id);
73      } catch (RemoteException e) {
74          e.printStackTrace();
75      }
76  }
77
78  /**
79   * Saves or adds the category
80   *
81   * @param data
82   */
83  public void saveOrUpdateCategory(MenuCategory data) {
84      try {
```

```
85     service.saveOrUpdateCategory(data);
86 } catch (RemoteException e) {
87     e.printStackTrace();
88 }
89 }
90
91 /**
92  * Saves or adds the menu
93  *
94  * @param data
95  */
96 public void saveOrUpdateMenu(Menu data) {
97     try {
98         service.saveOrUpdateMenu(data);
99     } catch (RemoteException e) {
100         e.printStackTrace();
101     }
102 }
103
104 /**
105  * Fetches all menus
106  *
107  * @return
108  */
109 public Collection<Menu> getAllMenus() {
110     try {
111         return service.getAllMenus();
112     } catch (RemoteException e) {
113         e.printStackTrace();
114     }
115     return null;
116 }
117
118 }
```

business/OrderService.java

```
1 package at.lumetsnet.caas.business;
2
3 import java.rmi.RemoteException;
4 import java.util.Collection;
5
6 import at.lumetsnet.caas.model.Order;
7 import at.lumetsnet.caas.rmi.interfaces.RemoteOrderService;
8
9 /**
10  * Mock order business logic class Used as singleton, should be replaced with
11  * real logic class
12  *
```



```
13  * @author romanlum
14  *
15  */
16
17 public class OrderService extends RmiService<RemoteOrderService> {
18
19     private static OrderService instance = null;
20
21     private OrderService() {
22         super("CaasOrderService");
23     }
24
25     /**
26      * singleton instance
27      */
28     public static OrderService getInstance() {
29         if (instance == null) {
30             instance = new OrderService();
31         }
32         return instance;
33     }
34
35     /**
36      * Fetches the orders of today
37      *
38      * @return
39      */
40     public Collection<Order> getTodaysOrders() {
41         try {
42             return service.getTodaysOrders();
43         } catch (RemoteException e) {
44             e.printStackTrace();
45         }
46         return null;
47     }
48 }
```

business/RmiService.java

```
1 package at.lumetsnet.caas.business;
2
3 import java.rmi.Naming;
4 import java.rmi.Remote;
5
6 /**
7  * Base class for all services using RMI
8  *
9  * @author romanlum
10  */
```

```
11  */
12  public class RmiService<T extends Remote> {
13
14      protected T service;
15      protected String serviceHost;
16      protected String name;
17
18      /**
19       * Create a service with the given name
20       *
21       * @param name
22       */
23      public RmiService(String name) {
24          this.name = name;
25      }
26
27      /**
28       * Initializes the service
29       *
30       * @param serviceHost
31       * @return
32       */
33      public boolean initialize(String serviceHost) {
34          try {
35              service = (T) Naming.lookup("rmi://" + serviceHost + "/" + name);
36          } catch (Exception e) {
37              e.printStackTrace();
38              return false;
39          }
40          return true;
41      }
42
43  }
```

business/UserService.java

```
1  package at.lumetsnet.caas.business;
2
3  import java.rmi.RemoteException;
4  import java.util.Collection;
5
6  import at.lumetsnet.caas.model.User;
7  import at.lumetsnet.caas.rmi.interfaces.RemoteUserService;
8
9  /**
10   * Mock user business logic class Used as singleton, should be replaced with
11   * real logic class
12   *
13   * @author romanlum
```

```
14  *
15  */
16  public class UserService extends RmiService<RemoteUserService> {
17
18      private static UserService instance = null;
19
20      private UserService() {
21          super("CaasUserService");
22      }
23
24      /**
25       * Singleton instance class
26       */
27      public static UserService getInstance() {
28          if (instance == null) {
29              instance = new UserService();
30          }
31          return instance;
32      }
33
34      /**
35       * Fetches all users
36       *
37       * @return
38       */
39      public Collection<User> getAllUsers() {
40          try {
41              return service.getAllUsers();
42          } catch (RemoteException e) {
43              e.printStackTrace();
44          }
45          return null;
46      }
47
48      /**
49       * delete the user
50       *
51       * @param id
52       */
53      public void deleteUser(long id) {
54          try {
55              service.deleteUser(id);
56          } catch (RemoteException e) {
57              e.printStackTrace();
58          }
59      }
60
61      /**
62       * Toggles the locked state of a user
```

```
63  *
64  * @param id
65  */
66  public void toggleLockState(long id) {
67      try {
68          service.toggleLockState(id);
69      } catch (RemoteException e) {
70          e.printStackTrace();
71      }
72  }
73
74  /**
75   * Saves or adds the user
76   *
77   * @param userModel
78   */
79  public void saveOrUpdate(User userModel) {
80      try {
81          service.saveOrUpdate(userModel);
82      } catch (RemoteException e) {
83          e.printStackTrace();
84      }
85  }
86  }
```

dal/DaoUtil.java

```
1  package at.lumetsnet.caas.dal;
2
3  import java.beans.BeanInfo;
4  import java.beans.Introspector;
5  import java.beans.PropertyDescriptor;
6  import java.sql.Connection;
7  import java.sql.Date;
8  import java.sql.PreparedStatement;
9  import java.sql.Statement;
10 import java.time.LocalDate;
11 import java.time.LocalDateTime;
12 import java.time.ZoneId;
13 import java.util.Arrays;
14 import java.util.Collection;
15
16 import at.lumetsnet.caas.model.Entity;
17
18 public class DaoUtil {
19
20     /**
21      * Generates a prepared statement for inserting an entity Sets all the
22      * values
```

```
23  *
24  * @param con
25  * @param entity
26  * @param tableName
27  * @param filter
28  * @return
29  */
30  public static PreparedStatement generateInsertStatement(Connection con,
31      Entity entity, String tableName, Collection<String> filter) {
32
33      BeanInfo info;
34      try {
35          info = Introspector.getBeanInfo(entity.getClass());
36          PropertyDescriptor[] pds = info.getPropertyDescriptors();
37
38          StringBuilder builder = new StringBuilder();
39          // build the insert string
40          builder.append(String.format("insert into '%s' (" , tableName));
41          Arrays.stream(pds).forEach(
42              x -> {
43                  if (!filter.stream().anyMatch(
44                      filterItem -> filterItem.equalsIgnoreCase(x
45                          .getName())) {
46                      builder.append(x.getName());
47                      builder.append(",");
48                  }
49              });
50          builder.delete(builder.length() - 1, builder.length()); // delete
51              // last ", "
52          builder.append(") VALUES (");
53          for (int i = 0; i < pds.length - filter.size(); i++) {
54              builder.append("?,");
55          }
56          builder.delete(builder.length() - 1, builder.length()); // delete
57              // last ", "
58          builder.append(")");
59
60          // create the statement
61          PreparedStatement stmt = con.prepareStatement(builder.toString(),
62              Statement.RETURN_GENERATED_KEYS);
63          int index = 1;
64          for (int i = 0; i < pds.length; i++) {
65              final String name = pds[i].getName();
66              // filter properties which should not be persisted
67              // lazy loading
68              if (!filter.stream().anyMatch(
69                  filterItem -> filterItem.equalsIgnoreCase(name))) {
70                  Object data = pds[i].getReadMethod().invoke(entity);
71                  // Handle LocalDate
```

```
72         if (data instanceof LocalDate) {
73             LocalDate date = (LocalDate) data;
74             stmt.setObject(
75                 index,
76                 Date.from(date.atStartOfDay()
77                     .atZone(ZoneId.systemDefault())
78                     .toInstant()));
79         } else if (data instanceof LocalDateTime) {
80             // handle localDateTime
81             LocalDateTime date = (LocalDateTime) data;
82             stmt.setObject(index, Date.from(date.atZone(
83                 ZoneId.systemDefault()).toInstant()));
84         } else
85             stmt.setObject(index, data);
86         index++;
87     }
88 }
89 return stmt;
90
91 } catch (Exception e) {
92     throw new DataAccessException(e.getMessage());
93 }
94
95 }
96
97 /**
98  * Creates an update statement for an entity
99  *
100  * @param con
101  * @param entity
102  * @param tableName
103  * @param filter
104  * @return
105  */
106 public static PreparedStatement generateUpdateStatement(Connection con,
107     Entity entity, String tableName, Collection<String> filter) {
108
109     BeanInfo info;
110     try {
111         info = Introspector.getBeanInfo(entity.getClass());
112         PropertyDescriptor[] pds = info.getPropertyDescriptors();
113
114         StringBuilder builder = new StringBuilder();
115         // build statement
116         builder.append(String.format("UPDATE '%s' SET ", tableName));
117         Arrays.stream(pds).forEach(
118             x -> {
119                 if (!filter.stream().anyMatch(
120                     filterItem -> filterItem.equalsIgnoreCase(x
```

```
121         .getName())) {
122             builder.append(x.getName() + " = ?");
123             builder.append(",");
124         }
125     });
126     builder.delete(builder.length() - 1, builder.length()); // delete
127                                     // last ","
128     // append id
129     builder.append(" WHERE id = ?");
130
131     // create statement
132     PreparedStatement stmt = con.prepareStatement(builder.toString(),
133     Statement.RETURN_GENERATED_KEYS);
134     int index = 1;
135     for (int i = 0; i < pds.length; i++) {
136         final String name = pds[i].getName();
137         if (!filter.stream().anyMatch(
138             filterItem -> filterItem.equalsIgnoreCase(name))) {
139             Object data = pds[i].getReadMethod().invoke(entity);
140             // Handle LocalDate
141             if (data instanceof LocalDate) {
142                 LocalDate date = (LocalDate) data;
143                 stmt.setObject(
144                     index,
145                     Date.from(date.atStartOfDay()
146                         .atZone(ZoneId.systemDefault())
147                         .toInstant()));
148             } else if (data instanceof LocalDateTime) {
149                 // Handle LocalDateTime
150                 LocalDateTime date = (LocalDateTime) data;
151                 stmt.setObject(index, Date.from(date.atZone(
152                     ZoneId.systemDefault()).toInstant()));
153             } else
154                 stmt.setObject(index, data);
155             index++;
156         }
157     }
158     stmt.setLong(index, entity.getId());
159     return stmt;
160
161 } catch (Exception e) {
162     throw new DataAccessException(e.getMessage());
163 }
164
165 }
166
167 }
```

dal/DataAccessException.java

```
1 package at.lumetsnet.caas.dal;
2
3 /**
4  * Generic dao exception
5  *
6  * @author romanlum
7  *
8  */
9 @SuppressWarnings("serial")
10 public class DataAccessException extends RuntimeException {
11     public DataAccessException(String msg) {
12         super(msg);
13     }
14 }
```

dal/GenericDao.java

```
1 package at.lumetsnet.caas.dal;
2
3 import java.util.Collection;
4
5 import at.lumetsnet.caas.model.Entity;
6
7 public interface GenericDao<T extends Entity> {
8
9     T get(long id);
10
11     Collection<T> getAll();
12
13     void saveOrUpdate(T entity);
14
15     void delete(long id);
16 }
```

dal/GenericJdbcDao.java

```
1 package at.lumetsnet.caas.dal;
2
3 import java.beans.BeanInfo;
4 import java.beans.IntrospectionException;
5 import java.beans.Introspector;
6 import java.beans.PropertyDescriptor;
7 import java.io.Closeable;
8 import java.io.IOException;
9 import java.sql.Connection;
10 import java.sql.Date;
11 import java.sql.DriverManager;
```



```
12 import java.sql.PreparedStatement;
13 import java.sql.ResultSet;
14 import java.sql.SQLException;
15 import java.sql.Timestamp;
16 import java.sql.Types;
17 import java.time.Instant;
18 import java.time.LocalDateTime;
19 import java.time.ZoneId;
20 import java.util.ArrayList;
21 import java.util.Arrays;
22 import java.util.Collection;
23
24 import at.lumetsnet.caas.model.Entity;
25
26 /**
27  * Generic dao for JDBC
28  *
29  * @author romanlum
30  *
31  * @param <T>
32  */
33 public abstract class GenericJdbcDao<T extends Entity> implements Closeable,
34     GenericDao<T> {
35
36     protected Connection con;
37     protected String conString;
38     protected String userName;
39     protected String password;
40     protected String tableName;
41     protected Class<T> entityClass;
42
43     public GenericJdbcDao(Class<T> entityClass, String tableName,
44         String conString, String userName, String password) {
45         this.conString = conString;
46         this.userName = userName;
47         this.password = password;
48         this.tableName = tableName;
49         this.entityClass = entityClass;
50
51     }
52
53     /**
54      * Gets and opens the connection
55      *
56      * @return
57      * @throws DataAccessException
58      */
59     public Connection getConnection() throws DataAccessException {
60         try {
```

```
61     if (con == null) {
62         con = DriverManager
63             .getConnection(conString, userName, password);
64     }
65     return con;
66 } catch (SQLException ex) {
67     throw new DataAccessException(ex.getMessage());
68 }
69 }
70
71 /**
72  * Closes the connection
73  */
74 @Override
75 public void close() throws IOException {
76     if (con != null) {
77         try {
78             con.close();
79             con = null;
80         } catch (SQLException e) {
81             throw new DataAccessException(e.getMessage());
82         }
83     }
84 }
85
86 /**
87  * Inserts or updates an entity based on the id ID == -1 means new entity
88  * Auto generated id is set after inserting
89  */
90 @Override
91 public void saveOrUpdate(T entity) {
92     if (entity.getId() == -1) {
93         try (PreparedStatement stmt = DaoUtil.generateInsertStatement(
94             getConnection(), entity, tableName, getPropertyFilter())) {
95             stmt.executeUpdate();
96             try (ResultSet rs = stmt.getGeneratedKeys()) {
97                 if (rs != null && rs.next()) {
98                     entity.setId(rs.getInt(1));
99                 } else {
100                     throw new DataAccessException(
101                         "Autogenerated keys are not supported by db");
102                 }
103             }
104         } catch (Exception ex) {
105             throw new DataAccessException(ex.getMessage());
106         }
107     } else {
108         try (PreparedStatement stmt = DaoUtil.generateUpdateStatement(
109             getConnection(), entity, tableName, getPropertyFilter())) {
```

```
110         stmt.executeUpdate();
111     } catch (Exception ex) {
112         throw new DataAccessException(ex.getMessage());
113     }
114
115 }
116 }
117
118 /**
119  * Gets a list of entities by using a where statement
120  *
121  * @param query
122  * @param args
123  * @return
124  * @throws DataAccessException
125  */
126 protected Collection<T> getFromWhere(String query, Object... args)
127     throws DataAccessException {
128     Collection<T> c = new ArrayList<T>();
129
130     BeanInfo info;
131     try {
132         info = Introspector.getBeanInfo(entityClass);
133     } catch (IntrospectionException e) {
134         throw new DataAccessException(e.getMessage());
135     }
136
137     PropertyDescriptor[] pds = info.getPropertyDescriptors();
138
139     // build statement
140     String sql = String.format("Select * from '%s' %s", tableName, query);
141
142     // / create statement
143     try (PreparedStatement pstmt = getConnection().prepareStatement(sql)) {
144         for (int i = 0; i < args.length; i++) {
145             pstmt.setObject(i + 1, args[i]);
146         }
147
148         try (ResultSet rs = pstmt.executeQuery()) {
149             while (rs.next()) {
150                 // create a new instance
151                 T entity = entityClass.newInstance();
152
153                 // fetch the meta data
154                 java.sql.ResultSetMetaData metaData = rs.getMetaData();
155                 for (int i = 1; i <= metaData.getColumnCount(); i++) {
156                     Object data = rs.getObject(i);
157
158                     // skip null values because they are already
```

```
159         // null in the new object
160         if (data == null)
161             continue;
162
163         String columnName = metaData.getColumnNames(i);
164         PropertyDescriptor descriptor = Arrays.stream(pds)
165             .filter(x -> x.getName().equals(columnName))
166             .findFirst().orElse(null);
167         if (descriptor != null) {
168             if (metaData.getColumnType(i) == Types.DATE) {
169                 // Date handling
170                 Instant instant = Instant
171                     .ofEpochMilli(((Date) data).getTime());
172                 data = LocalDateTime.ofInstant(instant,
173                     ZoneId.systemDefault()).toLocalDate();
174             } else if (metaData.getColumnType(i) == Types.TIMESTAMP) {
175                 // Time handling
176                 Instant instant = Instant
177                     .ofEpochMilli(((Timestamp) data)
178                         .getTime());
179                 data = LocalDateTime.ofInstant(instant,
180                     ZoneId.systemDefault());
181             }
182             // set the property
183             descriptor.getWriteMethod().invoke(entity, data);
184         }
185     }
186     }
187     c.add(entity);
188 }
189 }
190 return c;
191
192 } catch (Exception ex) {
193     throw new DataAccessException(ex.getMessage());
194 }
195 }
196
197 /**
198  * Gets the entity by id
199  */
200 @Override
201 public T get(long id) {
202     return getFromWhere("where id = ?", id).stream().findFirst()
203         .orElse(null);
204 }
205
206 /**
207  * Gets all entites
```

```
208     */
209     @Override
210     public Collection<T> getAll() {
211         return getFromWhere("");
212     }
213
214     /**
215      * Deletes an entity
216      */
217     @Override
218     public void delete(long id) {
219         try (PreparedStatement pstmt = getConnection().prepareStatement(
220             String.format("DELETE FROM '%s' where id = ?", tableName))) {
221             pstmt.setLong(1, id);
222             pstmt.executeUpdate();
223         } catch (SQLException ex) {
224             throw new DataAccessException(ex.getMessage());
225         }
226     }
227
228     /**
229      * Property filter used for insert and update statements These properties
230      * are skipped
231      *
232      * @return
233      */
234     public Collection<String> getPropertyFilter() {
235         ArrayList<String> filter = new ArrayList<>();
236         filter.add("class");
237         filter.add("id");
238         return filter;
239     }
240 }
241
242 }
```

dal/MenuCategoryDao.java

```
1 package at.lumetsnet.caas.dal;
2
3 import at.lumetsnet.caas.model.MenuCategory;
4
5 /**
6  * MenuCategory dao
7  *
8  * @author romanlum
9  *
10  */
11 public interface MenuCategoryDao extends GenericDao<MenuCategory> {
```

12
13 }

dal/MenuCategoryDaoJdbc.java

```
1 package at.lumetsnet.caas.dal;
2
3 import at.lumetsnet.caas.model.MenuCategory;
4
5 /**
6  * MenuCategory dao jdbc impl
7  *
8  * @author romanlum
9  *
10 */
11 public class MenuCategoryDaoJdbc extends GenericJdbcDao<MenuCategory> implements
12     MenuCategoryDao {
13
14     public MenuCategoryDaoJdbc(String conString, String userName,
15         String password) {
16         super(MenuCategory.class, "MenuCategory", conString, userName, password);
17     }
18
19 }
```

dal/MenuDao.java

```
1 package at.lumetsnet.caas.dal;
2
3 import at.lumetsnet.caas.model.Menu;
4
5 /**
6  * Menu dao
7  *
8  * @author romanlum
9  *
10 */
11
12 public interface MenuDao extends GenericDao<Menu> {
13
14 }
```

dal/MenuDaoJdbc.java

```
1 package at.lumetsnet.caas.dal;
2
3 import java.util.Collection;
```

```
4
5 import at.lumetsnet.caas.model.Menu;
6
7 /**
8  * Menu dao jdbc impl
9  *
10  * @author romanlum
11  *
12  */
13
14 public class MenuDaoJdbc extends GenericJdbcDao<Menu> implements MenuDao {
15
16     public MenuDaoJdbc(String conString, String userName, String password) {
17         super(Menu.class, "Menu", conString, userName, password);
18     }
19
20     @Override
21     public Collection<String> getPropertyFilter() {
22         Collection<String> result = super.getPropertyFilter();
23         result.add("category");
24         return result;
25     }
26
27 }
```

dal/OrderDao.java

```
1 package at.lumetsnet.caas.dal;
2
3 import java.time.LocalDate;
4 import java.util.Collection;
5
6 import at.lumetsnet.caas.model.Order;
7
8 /**
9  * Order dao
10  *
11  * @author romanlum
12  *
13  */
14
15 public interface OrderDao extends GenericDao<Order> {
16
17     /**
18      * Gets the orders by date
19      *
20      * @param date
21      * @return
22      */
23 }
```

```
23 Collection<Order> getOrdersByDate(LocalDate date);
24 }
```

dal/OrderDaoJdbc.java

```
1 package at.lumetsnet.caas.dal;
2
3 import java.sql.Date;
4 import java.time.LocalDate;
5 import java.util.Collection;
6
7 import at.lumetsnet.caas.model.Order;
8
9 /**
10  * Order dao jdbc impl
11  *
12  * @author romanlum
13  *
14  */
15 public class OrderDaoJdbc extends GenericJdbcDao<Order> implements OrderDao {
16
17     public OrderDaoJdbc(String conString, String userName, String password) {
18         super(Order.class, "Order", conString, userName, password);
19     }
20
21     /**
22      * Gets the orders by date
23      */
24     public Collection<Order> getOrdersByDate(LocalDate date) {
25         return getFromWhere("WHERE DATE(time) = ?", Date.valueOf(date));
26     }
27
28     /**
29      * Sets property filters to filter Referenced entities
30      */
31     @Override
32     public Collection<String> getPropertyFilter() {
33         Collection<String> result = super.getPropertyFilter();
34         result.add("user");
35         result.add("menu");
36         return result;
37     }
38
39 }
```

dal/UserDao.java

```
1 package at.lumetsnet.caas.dal;
2
3 import at.lumetsnet.caas.model.User;
4
5 /**
6  * User dao
7  *
8  * @author romanlum
9  *
10 */
11 public interface UserDao extends GenericDao<User> {
12
13 }
```

dal/UserDaoJdbc.java

```
1 package at.lumetsnet.caas.dal;
2
3 import at.lumetsnet.caas.model.User;
4
5 /**
6  * User dao jdbc impl
7  *
8  * @author romanlum
9  *
10 */
11 public class UserDaoJdbc extends GenericJdbcDao<User> implements UserDao {
12
13     public UserDaoJdbc(String conString, String userName, String password) {
14         super(User.class, "User", conString, userName, password);
15     }
16
17 }
```

gui/ActionTableCell.java

```
1 package at.lumetsnet.caas.gui;
2
3 import java.util.function.Consumer;
4
5 import javafx.scene.control.Button;
6 import javafx.scene.control.TableCell;
7 import javafx.scene.layout.HBox;
8
9 /**
10  * TableCell used for displaying edit / delete buttons inside a TableView
11  *
12  * @author romanlum
```

```
13  *
14  * @param <S>
15  * @param <T>
16  */
17 public class ActionTableCell<S, T> extends TableCell<S, T> {
18     protected HBox box = new HBox();
19     protected Button editButton = new Button("Bearbeiten");
20     protected Button deleteButton = new Button("Löschen");
21
22     protected Consumer<T> editAction;
23     protected Consumer<T> deleteAction;
24
25     public ActionTableCell(Consumer<T> editAction, Consumer<T> deleteAction) {
26         this.editAction = editAction;
27         this.deleteAction = deleteAction;
28
29         box.getStyleClass().add("table-command-container");
30         editButton.getStyleClass()
31             .addAll("table-command", "table-command-edit");
32         deleteButton.getStyleClass().addAll("table-command",
33             "table-command-delete");
34         box.getChildren().addAll(editButton, deleteButton);
35     }
36
37     @Override
38     protected void updateItem(T entity, boolean empty) {
39
40         if (entity != null) {
41             editButton.setOnAction(x -> {
42                 if (editAction != null)
43                     editAction.accept(entity);
44             });
45             deleteButton.setOnAction(x -> {
46                 if (deleteAction != null)
47                     deleteAction.accept(entity);
48             });
49             setGraphic(box);
50         } else {
51             setGraphic(null);
52         }
53     }
54 }
55 }
```

gui/AmountTableCell.java

```
1 package at.lumetsnet.caas.gui;
2
3 import javafx.scene.control.TableCell;
```

```
4
5 /**
6  * TableCell used for formatting an amount inside a TableView Formats the value
7  * given in LOWEST currency unit to normal EUR representation
8  *
9  * @author romanlum
10 *
11 * @param <S>
12 */
13 public class AmountTableCell<S> extends TableCell<S, Number> {
14
15     @Override
16     protected void updateItem(Number entity, boolean empty) {
17
18         if (entity != null) {
19             // formats the value given in lowest currency unit
20             // to EUR
21             setText(((double) ((long) entity) / 100) + " EUR");
22         } else {
23             setText("");
24         }
25     }
26 }
```

gui/DateTableCell.java

```
1 package at.lumetsnet.caas.gui;
2
3 import java.time.LocalDate;
4 import java.time.format.DateTimeFormatter;
5
6 import javafx.scene.control.TableCell;
7
8 /**
9  * TableCell used for displaying LocalDate values inside a TableView
10 *
11 * @author romanlum
12 *
13 * @param <S>
14 */
15 public class DateTableCell<S> extends TableCell<S, LocalDate> {
16
17     @Override
18     protected void updateItem(LocalDate entity, boolean empty) {
19
20         if (entity != null) {
21             setText(entity.format(DateTimeFormatter.ISO_LOCAL_DATE));
22         } else {
23             setText("");
24         }
25     }
26 }
```

```
24         setText("");
25     }
26 }
27 }
```

gui/MainWindow.java

```
1 package at.lumetsnet.caas.gui;
2
3 import java.util.HashMap;
4
5 import javafx.scene.Node;
6 import javafx.scene.Scene;
7 import javafx.scene.control.Button;
8 import javafx.scene.control.Label;
9 import javafx.scene.layout.BorderPane;
10 import javafx.scene.layout.HBox;
11 import javafx.scene.layout.Pane;
12 import javafx.scene.layout.Priority;
13 import javafx.scene.layout.VBox;
14 import javafx.stage.Stage;
15 import javafx.stage.Window;
16 import at.lumetsnet.caas.gui.pages.ManageMenusPage;
17 import at.lumetsnet.caas.gui.pages.ManageUsersPage;
18 import at.lumetsnet.caas.gui.pages.OrdersPage;
19 import at.lumetsnet.caas.gui.pages.Showable;
20 import at.lumetsnet.caas.model.User;
21
22 /**
23  * Main windows class Creates all pages and manages page switching
24  *
25  * @author romanlum
26  *
27  */
28 public class MainWindow {
29
30     private enum VIEW_TYPE {
31         ADMIN_ORDERS, ADMIN_USERS, ADMIN_MENUS
32     }
33
34     private User user;
35     private Stage stage = new Stage();
36     private HashMap<VIEW_TYPE, Pane> pages;
37     private BorderPane container;
38
39     private Button adminOrdersViewButton;
40     private Button adminUsersViewButton;
41     private Button adminMenusViewButton;
42 }
```

```
43 public MainWindow(Window owner, User user) {
44     this.user = user;
45     container = new BorderPane();
46     container.getStyleClass().add("window-container");
47     container.setTop(createTopMenu());
48
49     pages = createPages();
50
51     Scene scene = new Scene(container);
52     scene.getStylesheets().add(
53         getClass().getResource("css/main-window.css").toExternalForm());
54
55     stage.initOwner(owner);
56     stage.setScene(scene);
57     stage.setTitle("CaaS");
58     // start with 1024/768
59     stage.setWidth(1024);
60     stage.setHeight(768);
61     switchView(VIEW_TYPE.ADMIN_ORDERS);
62
63 }
64
65 private HashMap<VIEW_TYPE, Pane> createPages() {
66     HashMap<VIEW_TYPE, Pane> pages = new HashMap<>();
67     pages.put(VIEW_TYPE.ADMIN_ORDERS, new OrdersPage());
68     pages.put(VIEW_TYPE.ADMIN_USERS, new ManageUsersPage());
69     pages.put(VIEW_TYPE.ADMIN_MENUS, new ManageMenusPage());
70
71     return pages;
72 }
73
74 public void show() {
75
76     stage.show();
77 }
78
79 private Node createTopMenu() {
80     HBox topContainer = new HBox();
81     topContainer.getStyleClass().add("nav-container");
82
83     Label iconLabel = new Label();
84     iconLabel.setId("nav-top-icon");
85     VBox descBox = new VBox();
86
87     Label userNameLabel = new Label();
88     userNameLabel.getStyleClass().add("nav-username");
89     userNameLabel.setText(user.getFirstName() + " " + user.getLastName());
90
91     Label descLabel = new Label();
```

```
192     descLabel.getStyleClass().add("nav-role");
193     descLabel.setText("Administrator");
194
195     descBox.getChildren().add(userNameLabel);
196     descBox.getChildren().add(descLabel);
197
198     HBox commandBox = new HBox();
199     commandBox.getStyleClass().add("nav-command-container");
200
201     adminOrdersViewButton = new Button();
202     adminOrdersViewButton.getStyleClass().add("nav-command");
203     adminOrdersViewButton.setText("Bestellungen");
204     adminOrdersViewButton
205         .setOnAction(x -> switchView(VIEW_TYPE.ADMIN_ORDERS));
206     commandBox.getChildren().add(adminOrdersViewButton);
207
208     adminUsersViewButton = new Button();
209     adminUsersViewButton.getStyleClass().add("nav-command");
210     adminUsersViewButton.setText("Benutzerverwaltung");
211     adminUsersViewButton
212         .setOnAction(x -> switchView(VIEW_TYPE.ADMIN_USERS));
213     commandBox.getChildren().add(adminUsersViewButton);
214
215     adminMenusViewButton = new Button();
216     adminMenusViewButton.getStyleClass().add("nav-command");
217     adminMenusViewButton.setText("Speisekarte verwalten");
218     adminMenusViewButton
219         .setOnAction(x -> switchView(VIEW_TYPE.ADMIN_MENUS));
220     commandBox.getChildren().add(adminMenusViewButton);
221
222     HBox.setHgrow(commandBox, Priority.ALWAYS);
223
224     topContainer.getChildren().add(iconLabel);
225     topContainer.getChildren().add(descBox);
226     topContainer.getChildren().add(commandBox);
227     return topContainer;
228 }
229
230 /**
231  * Switches the view
232  *
233  * @param type
234  */
235 private void switchView(VIEW_TYPE type) {
236     adminOrdersViewButton.getStyleClass().remove("nav-command-selected");
237     adminUsersViewButton.getStyleClass().remove("nav-command-selected");
238     adminMenusViewButton.getStyleClass().remove("nav-command-selected");
239
240     Button selected = null;
```

```
141     switch (type) {
142     case ADMIN_USERS:
143         selected = adminUsersViewButton;
144
145         break;
146     case ADMIN_ORDERS:
147         selected = adminOrdersViewButton;
148         break;
149     case ADMIN_MENUS:
150         selected = adminMenusViewButton;
151         break;
152     }
153     if (selected != null)
154         selected.getStyleClass().add("nav-command-selected");
155
156     Pane page = pages.get(type);
157     if (page instanceof Showable)
158         ((Showable) page).show();
159     container.setCenter(page);
160
161 }
162
163 }
```

gui/ManageUserActionCell.java

```
1 package at.lumetsnet.caas.gui;
2
3 import java.util.function.Consumer;
4
5 import javafx.scene.control.Button;
6 import at.lumetsnet.caas.viewmodel.UserViewModel;
7
8 /**
9  * TableCell used for displaying additional lock/unlock button inside the
10  * TableView
11  *
12  * @author romanlum
13  *
14  * @param <S>
15  * @param <T>
16  */
17 public class ManageUserActionCell<S, T> extends ActionTableCell<S, T> {
18
19     protected Button lockButton = new Button("Sperren");
20     protected Consumer<T> lockAction;
21
22     public ManageUserActionCell(Consumer<T> editAction,
23         Consumer<T> deleteAction, Consumer<T> lockAction) {
```

```
24     super(editAction, deleteAction);
25     this.lockAction = lockAction;
26     box.getChildren().addAll(lockButton);
27 }
28
29 @Override
30 protected void updateItem(T entity, boolean empty) {
31
32     if (entity != null) {
33         lockButton.setOnAction(x -> {
34             if (lockAction != null)
35                 lockAction.accept(entity);
36
37         });
38         if (getTableRow() != null && getTableRow().getItem() != null) {
39             Object model = getTableRow().getItem();
40             if (model instanceof UserViewModel) {
41                 lockButton.getStyleClass().removeIf(
42                     x -> x.startsWith("table-command"));
43                 if (((UserViewModel) model).getLockedProperty().get()) {
44                     lockButton.setText("Entsperren");
45                     lockButton.getStyleClass().addAll("table-command",
46                                                         "table-command-unlock");
47                 } else {
48                     lockButton.setText("Sperren");
49                     lockButton.getStyleClass().addAll("table-command",
50                                                         "table-command-lock");
51                 }
52             }
53         }
54     }
55     super.updateItem(entity, empty);
56 }
57
58
59 }
```

gui/Util.java

```
1 package at.lumetsnet.caas.gui;
2
3 import java.util.ArrayList;
4 import java.util.Collection;
5
6 import javafx.beans.property.ObjectProperty;
7 import javafx.beans.property.StringProperty;
8 import javafx.beans.value.ChangeListener;
9 import javafx.beans.value.ObservableValue;
10 import javafx.collections.ObservableList;
```



```
11 import javafx.scene.Node;
12 import javafx.scene.control.ComboBox;
13 import javafx.scene.control.Label;
14 import javafx.scene.control.PasswordField;
15 import javafx.scene.control.TextField;
16
17 /**
18  * Common gui utils
19  *
20  * @author romanlum
21  *
22  */
23 public class Util {
24
25     /**
26      * Creates a form label
27      *
28      * @param text
29      * @return
30      */
31     public static Node getFormLabel(String text) {
32         Label lab = new Label();
33         lab.setText(text);
34         lab.getStyleClass().add("form-label");
35         return lab;
36     }
37
38     /**
39      * Creates a text field form and label
40      *
41      * @param text
42      * @param property
43      * @return
44      */
45     public static Collection<Node> getTextFieldForm(String text,
46         StringProperty property) {
47
48         Collection<Node> nodes = new ArrayList<Node>();
49         nodes.add(getFormLabel(text));
50
51         TextField field = new TextField();
52         field.getStyleClass().add("form-textfield");
53         field.textProperty().bindBidirectional(property);
54         nodes.add(field);
55         return nodes;
56     }
57
58     /**
59      * Creates a password form field and label
```

```
60  *
61  * @param text
62  * @param property
63  * @return
64  */
65  public static Collection<Node> getPasswordFieldForm(String text,
66      StringProperty property) {
67
68      Collection<Node> nodes = new ArrayList<Node>();
69      nodes.add(getFormLabel(text));
70
71      PasswordField field = new PasswordField();
72      field.getStyleClass().add("form-textfield");
73      field.textProperty().bindBidirectional(property);
74      nodes.add(field);
75      return nodes;
76  }
77
78  /**
79   * Creates a combo box form and label
80   *
81   * @param text
82   * @param property
83   * @param data
84   * @return
85   */
86  public static <T> Collection<Node> getComboboxForm(String text,
87      ObjectProperty<T> property, ObservableList<T> data) {
88
89      Collection<Node> nodes = new ArrayList<Node>();
90      nodes.add(getFormLabel(text));
91
92      ComboBox<T> field = new ComboBox<>();
93      field.getStyleClass().add("form-combobox");
94      field.setItems(data);
95      field.getSelectionModel().selectedItemProperty()
96          .addListener(new ChangeListener<T>() {
97              @Override
98              public void changed(ObservableValue<? extends T> item,
99                  T arg1, T arg2) {
100                  property.setValue(item.getValue());
101              }
102          });
103      field.selectionModelProperty().get().select(property.get());
104      nodes.add(field);
105      return nodes;
106  }
107  }
108
```

109 }

gui/dialogs/Dialog.java

```
1 package at.lumetsnet.caas.gui.dialogs;
2
3 import javafx.scene.Node;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Label;
6 import javafx.scene.layout.BorderPane;
7 import javafx.scene.layout.HBox;
8 import javafx.scene.layout.Pane;
9 import javafx.scene.layout.VBox;
10 import javafx.stage.Modality;
11 import javafx.stage.Stage;
12 import javafx.stage.StageStyle;
13 import javafx.stage.Window;
14
15 /**
16  * Dialog base class used for all dialogs
17  *
18  * @author romanlum
19  *
20  */
21 public abstract class Dialog {
22
23     protected Stage dialogStage = new Stage();
24
25     protected final void createGui(Window owner, String title,
26         String description) {
27         BorderPane layout = new BorderPane();
28         layout.getStyleClass().add("dialog-container");
29         layout.setTop(createTopPane(title, description));
30         layout.setCenter(createContentPane());
31         layout.setBottom(createBottomPane());
32
33         Scene dialogScene = new Scene(layout);
34         dialogScene.getStylesheets().add(
35             getClass().getResource("../css/dialog.css").toExternalForm());
36
37         dialogStage.initOwner(owner);
38         dialogStage.setScene(dialogScene);
39         dialogStage.setTitle(title);
40         dialogStage.initModality(Modality.WINDOW_MODAL);
41         dialogStage.initStyle(StageStyle.UTILITY);
42         dialogStage.setResizable(false);
43     }
44
45     protected Node createTopPane(String title, String description) {
```

```
46     HBox box = new HBox();
47     box.getStyleClass().add("dialog-top-container");
48
49     Label iconLabel = new Label();
50     iconLabel.setId("dialog-icon");
51     VBox descBox = new VBox();
52
53     Label titleLabel = new Label();
54     titleLabel.getStyleClass().add("dialog-title");
55     titleLabel.setText(title);
56
57     Label descLabel = new Label();
58     descLabel.getStyleClass().add("dialog-description");
59     descLabel.setText(description);
60
61     descBox.getChildren().add(titleLabel);
62     descBox.getChildren().add(descLabel);
63
64     box.getChildren().add(iconLabel);
65     box.getChildren().add(descBox);
66     return box;
67 }
68
69 protected Pane createBottomPane() {
70     HBox box = new HBox();
71     box.getStyleClass().add("dialog-bottom-container");
72     return box;
73 }
74 }
75
76 /**
77  * Content used for the dialog
78  *
79  * @return
80  */
81 protected abstract Node createContentPane();
82
83 }
```

gui/dialogs/AlertDialog.java

```
1 package at.lumetsnet.caas.gui.dialogs;
2
3 import javafx.scene.Node;
4 import javafx.scene.control.Button;
5 import javafx.scene.layout.Pane;
6 import javafx.stage.Window;
7
8 /**
```

```
9  * Own dialog class used because javafx does not have dialogs before update 40
10 * :(
11 *
12 * @author romanlum
13 *
14 */
15 public class ErrorDialog extends Dialog {
16
17     public ErrorDialog(Window owner) {
18
19         createGui(owner, "Fehler", "Bitte korrigieren Sie Ihre Eingaben.");
20
21         dialogStage
22             .getScene()
23             .getStylesheets()
24             .add(getClass().getResource("../css/error-dialog.css")
25                 .toExternalForm());
26     }
27
28     @Override
29     protected Pane createBottomPane() {
30         Pane pane = super.createBottomPane();
31         Button button = new Button();
32         button.setText("OK");
33         button.getStyleClass().add("command-button");
34         button.setOnAction(x -> okCommand());
35         pane.getChildren().add(button);
36         return pane;
37     }
38
39     private void okCommand() {
40         dialogStage.close();
41     }
42
43     @Override
44     protected Node createContentPane() {
45         return null;
46     }
47
48     public boolean show() {
49         dialogStage.showAndWait();
50         return true;
51     }
52
53     public static boolean show(Window owner) {
54         ErrorDialog dialog = new ErrorDialog(owner);
55         return dialog.show();
56     }
57 }
```

58 }

gui/dialogs/ManageEntityDialog.java

```
1 package at.lumetsnet.caas.gui.dialogs;
2
3 import javafx.beans.property.BooleanProperty;
4 import javafx.beans.property.SimpleBooleanProperty;
5 import javafx.scene.control.Button;
6 import javafx.scene.layout.Pane;
7 import at.lumetsnet.caas.viewmodel.Validatable;
8
9 /**
10  * Base dialog used for editing a viewmodel entity
11  *
12  * @author romanlum
13  *
14  * @param <T>
15  */
16 public abstract class ManageEntityDialog<T> extends Dialog {
17
18     protected boolean canceled = false;
19     protected T viewModel;
20     protected BooleanProperty validationErrorProperty = new SimpleBooleanProperty();
21
22     /**
23      * Shows the dialog and waits for exit returns if the action was canceled
24      *
25      * @return
26      */
27     public boolean show() {
28         dialogStage.showAndWait();
29         return canceled;
30     }
31
32     /**
33      * Save action
34      */
35     protected abstract void saveCommand();
36
37     /**
38      * Cancel action Default: closes the dialog and sets canceled flag
39      */
40     protected void cancelCommand() {
41         canceled = true;
42         dialogStage.close();
43     }
44 }
45
```

```
46  /**
47   * Validates the viewmodel if it is Validatable Shows error dialog if
48   * validation fails
49   *
50   * @return true if the viewmodel is not validatable true on successfull
51   *         validation otherwise false
52   */
53  protected boolean validate() {
54      if (!(viewModel instanceof Validatable)) {
55          return true;
56      }
57
58      boolean result = ((Validatable) viewModel).validate();
59      if (!result) {
60          ErrorDialog.show(dialogStage);
61      }
62      return result;
63  }
64
65  @Override
66  protected Pane createBottomPane() {
67      Pane pane = super.createBottomPane();
68      Button button = new Button();
69      button.setText("Speichern");
70      button.getStyleClass().add("command-button");
71      button.setOnAction(x -> saveCommand());
72      pane.getChildren().add(button);
73
74      button = new Button();
75      button.setText("Abbrechen");
76      button.getStyleClass().add("command-button");
77      button.setOnAction(x -> cancelCommand());
78      pane.getChildren().add(button);
79
80      return pane;
81  }
82
83 }
```

gui/dialogs/ManageMenuCategoryDialog.java

```
1  package at.lumetsnet.caas.gui.dialogs;
2
3  import javafx.scene.Node;
4  import javafx.scene.layout.VBox;
5  import javafx.stage.Window;
6  import at.lumetsnet.caas.business.MenuService;
7  import at.lumetsnet.caas.gui.Util;
8  import at.lumetsnet.caas.model.MenuCategory;
```

```
9 import at.lumetsnet.caas.viewmodel.MenuCategoryViewModel;
10
11 /**
12  * Edit/Add dialog for MenuCategories
13  *
14  * @author romanlum
15  *
16  */
17 public class ManageMenuCategoryDialog extends
18     ManageEntityDialog<MenuCategoryViewModel> {
19
20     public ManageMenuCategoryDialog(Window owner, MenuCategory data) {
21         viewModel = new MenuCategoryViewModel(data);
22         createGui(owner, "Bereich verwalten",
23             "Bitte geben Sie die Daten des Bereichs ein.");
24
25         dialogStage
26             .getScene()
27             .getStylesheets()
28             .add(getClass().getResource(
29                 "../css/manage-menu-category-dialog.css")
30                 .toExternalForm());
31     }
32
33     @Override
34     protected Node createContentPane() {
35
36         VBox box = new VBox();
37         box.getStyleClass().add("form-container");
38         box.getChildren().addAll(
39             Util.getTextFieldForm("Name", viewModel.getNameProperty()));
40
41         return box;
42     }
43
44     @Override
45     protected void saveCommand() {
46         if (validate()) {
47             // Saves the entity
48             MenuService.getInstance().saveOrUpdateCategory(
49                 viewModel.toCategoryModel());
50             dialogStage.close();
51         }
52     }
53 }
54
55 /**
56  * Static method for showing the dialog
57  *
```



```
58  * @param owner
59  * @param model
60  * @return
61  */
62  public static boolean show(Window owner, MenuCategory model) {
63      ManageMenuCategoryDialog dialog = new ManageMenuCategoryDialog(owner,
64          model);
65      return dialog.show();
66  }
67  }
68  }
```

gui/dialogs/ManageMenuDialog.java

```
1  package at.lumetsnet.caas.gui.dialogs;
2
3  import javafx.collections.FXCollections;
4  import javafx.collections.ObservableList;
5  import javafx.scene.Node;
6  import javafx.scene.control.DatePicker;
7  import javafx.scene.layout.HBox;
8  import javafx.scene.layout.VBox;
9  import javafx.stage.Window;
10 import at.lumetsnet.caas.business.MenuService;
11 import at.lumetsnet.caas.gui.Util;
12 import at.lumetsnet.caas.model.Menu;
13 import at.lumetsnet.caas.viewmodel.MenuCategoryViewModel;
14 import at.lumetsnet.caas.viewmodel.MenuViewModel;
15
16 /**
17  * Edit/Add dialog for menus
18  *
19  * @author romanlum
20  *
21  */
22 public class ManageMenuDialog extends ManageEntityDialog<MenuViewModel> {
23
24     public ManageMenuDialog(Window owner, Menu menu) {
25         viewModel = new MenuViewModel(menu);
26         createGui(owner, "Hauptspeise verwalten",
27             "Bitte geben Sie die Daten der Hauptspeise ein.");
28
29         dialogStage
30             .getScene()
31             .getStylesheets()
32             .add(getClass().getResource("../css/manage-menu-dialog.css"))
33             .toExternalForm());
34     }
35 }
```

```
36  @Override
37  protected Node createContentPane() {
38
39      VBox box = new VBox();
40      box.getStyleClass().add("form-container");
41      box.getChildren().addAll(
42          Util.getTextFieldForm("Beschreibung",
43              viewModel.getDescriptionProperty()));
44      box.getChildren().addAll(
45          Util.getComboboxForm("Bereich",
46              viewModel.getCategoryProperty(), getMenuCategories()));
47
48      box.getChildren().addAll(
49          Util.getTextFieldForm("Preis",
50              viewModel.getPriceAsStringProperty()));
51      DatePicker picker = new DatePicker();
52      box.getChildren().add(Util.getFormLabel("Zeitraum"));
53
54      HBox rangeBox = new HBox();
55      rangeBox.setSpacing(10);
56
57      picker.valueProperty().bindBidirectional(viewModel.getBeginProperty());
58      picker.getStyleClass().add("form-datepicker");
59      rangeBox.getChildren().add(picker);
60
61      picker = new DatePicker();
62      picker.getStyleClass().add("form-datepicker");
63      picker.valueProperty().bindBidirectional(viewModel.getEndProperty());
64      rangeBox.getChildren().add(picker);
65      box.getChildren().add(rangeBox);
66      return box;
67  }
68
69  private ObservableList<MenuCategoryViewModel> getMenuCategories() {
70      ObservableList<MenuCategoryViewModel> data = FXCollections
71          .observableArrayList();
72      MenuService.getInstance().getAllCategories()
73          .forEach(x -> data.add(new MenuCategoryViewModel(x)));
74      return data;
75  }
76
77  @Override
78  protected void saveCommand() {
79      if (validate()) {
80          // Saves the entity
81          MenuService.getInstance().saveOrUpdateMenu(viewModel.toMenuModel());
82          dialogStage.close();
83      }
84  }
```

```
85     }
86
87     /**
88      * Static method for showing the dialog
89      *
90      * @param owner
91      * @param model
92      * @return
93      */
94     public static boolean show(Window owner, Menu model) {
95         ManageMenuDialog dialog = new ManageMenuDialog(owner, model);
96         return dialog.show();
97     }
98 }
99 }
```

gui/dialogs/ManageUserDialog.java

```
1 package at.lumetsnet.caas.gui.dialogs;
2
3 import javafx.scene.Node;
4 import javafx.scene.layout.VBox;
5 import javafx.stage.Window;
6 import at.lumetsnet.caas.business.UserService;
7 import at.lumetsnet.caas.gui.Util;
8 import at.lumetsnet.caas.model.User;
9 import at.lumetsnet.caas.viewmodel.UserViewModel;
10
11 public class ManageUserDialog extends ManageEntityDialog<UserViewModel> {
12
13     public ManageUserDialog(Window owner, User user) {
14         viewModel = new UserViewModel(user);
15         createGui(owner, "User verwalten",
16             "Bitte geben Sie die Daten des Benutzers ein.");
17
18         dialogStage
19             .getScene()
20             .getStylesheets()
21             .add(getClass().getResource("../css/manage-user-dialog.css")
22                 .toExternalForm());
23     }
24
25     @Override
26     protected Node createContentPane() {
27
28         VBox box = new VBox();
29         box.getStyleClass().add("form-container");
30         box.getChildren().addAll(
31             Util.getTextFieldForm("Benutzername",
```

```
32         viewModel.getUserNameProperty());
33     box.getChildren().addAll(
34         Util.getPasswordFieldForm("Passwort",
35             viewModel.getPasswordProperty()));
36     box.getChildren().addAll(
37         Util.getTextFieldForm("Vorname",
38             viewModel.getFirstNameProperty()));
39     box.getChildren().addAll(
40         Util.getTextFieldForm("Nachname",
41             viewModel.getLastNameProperty()));
42
43     return box;
44 }
45
46 @Override
47 protected void saveCommand() {
48     if (validate()) {
49         // Saves the entity
50         UserService.getInstance().saveOrUpdate(viewModel.toUserModel());
51         dialogStage.close();
52     }
53 }
54
55 /**
56  * Static method for showing the dialog
57  *
58  * @param owner
59  * @param model
60  * @return
61  */
62 public static boolean show(Window owner, User model) {
63     ManageUserDialog dialog = new ManageUserDialog(owner, model);
64     return dialog.show();
65 }
66 }
67 }
```

gui/pages/ManageMenusPage.java

```
1 package at.lumetsnet.caas.gui.pages;
2
3 import java.util.Optional;
4
5 import javafx.scene.Node;
6 import javafx.scene.control.Button;
7 import javafx.scene.control.Label;
8 import javafx.scene.control.TableColumn;
9 import javafx.scene.control.TableView;
10 import javafx.scene.layout.HBox;
```

```
11 import javafx.scene.layout.Priority;
12 import javafx.scene.layout.VBox;
13 import at.lumetsnet.caas.gui.ActionTableCell;
14 import at.lumetsnet.caas.gui.AmountTableCell;
15 import at.lumetsnet.caas.gui.dialogs.ManageMenuCategoryDialog;
16 import at.lumetsnet.caas.gui.dialogs.ManageMenuDialog;
17 import at.lumetsnet.caas.viewmodel.ManageMenusPageViewModel;
18 import at.lumetsnet.caas.viewmodel.MenuCategoryViewModel;
19 import at.lumetsnet.caas.viewmodel.MenuViewModel;
20
21 /**
22  * View page used for managing menus Uses a ManageMenusPageViewModel for
23  * business logic operations
24  *
25  * @author romanlum
26  *
27  */
28 public class ManageMenusPage extends VBox implements Showable {
29
30     TableView<MenuCategoryViewModel> categoryTable;
31     TableView<MenuViewModel> menuTable;
32     ManageMenusPageViewModel viewModel;
33
34     public ManageMenusPage() {
35
36         viewModel = new ManageMenusPageViewModel();
37         getStyleClass().add("page-content-container");
38
39         HBox topPane = new HBox();
40         topPane.getChildren().add(createTitle("Bereiche"));
41         HBox commandContainer = new HBox();
42         commandContainer.getStyleClass().add("page-command-container");
43         commandContainer.getChildren().add(createAddCategoryButton());
44         HBox.setHgrow(commandContainer, Priority.ALWAYS);
45
46         topPane.getChildren().add(commandContainer);
47         categoryTable = createCategoryTableView();
48         categoryTable.setItems(viewModel.getCategoryList());
49
50         HBox menuTopPane = new HBox();
51         menuTopPane.getChildren().add(createTitle("Hauptspeisen"));
52         HBox menuCommandContainer = new HBox();
53         menuCommandContainer.getStyleClass().add("page-command-container");
54         menuCommandContainer.getChildren().add(createAddMenuButton());
55         HBox.setHgrow(menuCommandContainer, Priority.ALWAYS);
56
57         menuTopPane.getChildren().add(menuCommandContainer);
58         menuTable = createMenuTableView();
59         menuTable.setItems(viewModel.getMenuList());
```

```
60
61     getChildren().addAll(topPane, categoryTable, menuTopPane, menuTable);
62
63 }
64
65 private Node createAddCategoryButton() {
66     Button button = new Button("Bereich anlegen");
67     button.getStyleClass().add("page-command");
68     button.setOnAction(x -> addCategoryCommand());
69     return button;
70 }
71
72 private Node createAddMenuButton() {
73     Button button = new Button("Hauptspeise anlegen");
74     button.getStyleClass().add("page-command");
75     button.setOnAction(x -> addMenuCommand());
76     return button;
77 }
78
79 private Label createTitle(String title) {
80     Label titleLabel = new Label(title);
81     titleLabel.getStyleClass().add("page-title");
82     return titleLabel;
83 }
84
85 private TableView<MenuCategoryViewModel> createCategoryTableView() {
86     TableView<MenuCategoryViewModel> table = new TableView<>();
87     table.getStyleClass().add("table");
88     table.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
89
90     TableColumn<MenuCategoryViewModel, String> column = new TableColumn<>(
91         "Name");
92     column.setCellValueFactory(x -> x.getValue().getNameProperty());
93     table.getColumns().add(column);
94
95     TableColumn<MenuCategoryViewModel, Number> actionColumn = new TableColumn<>(
96         "Aktion");
97     actionColumn.setCellValueFactory(x -> x.getValue().getIdProperty());
98     actionColumn.setCellFactory(p -> new ActionTableCell<>(
99         x -> editCategoryCommand(x), x -> deleteCategoryCommand(x)));
100
101     table.getColumns().add(actionColumn);
102     return table;
103 }
104
105 private TableView<MenuViewModel> createMenuTableView() {
106     TableView<MenuViewModel> table = new TableView<>();
107     table.getStyleClass().add("table");
108     table.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
```

```
109
110     TableColumn<MenuViewModel, String> column = new TableColumn<>("Name");
111     column.setCellValueFactory(x -> x.getValue().getDescriptionProperty());
112     table.getColumns().add(column);
113
114     TableColumn<MenuViewModel, Number> priceColumn = new TableColumn<>(
115         "Preis");
116     priceColumn.setCellValueFactory(x -> x.getValue().getPriceProperty());
117     priceColumn.setCellFactory(x -> new AmountTableCell<MenuViewModel>());
118     table.getColumns().add(priceColumn);
119
120     column = new TableColumn<>("Zeitraum");
121     column.setCellValueFactory(x -> x.getValue().getUsageRangeProperty());
122     table.getColumns().add(column);
123
124     TableColumn<MenuViewModel, Number> actionColumn = new TableColumn<>(
125         "Aktion");
126     actionColumn.setCellValueFactory(x -> x.getValue().getIdProperty());
127     actionColumn.setCellFactory(p -> new ActionTableCell<>(
128         x -> editMenuCommand(x), x -> deleteMenuCommand(x)));
129
130     table.getColumns().add(actionColumn);
131     return table;
132 }
133
134 private void deleteCategoryCommand(Number x) {
135     viewModel.deleteCategoryCommand(x);
136 }
137
138
139 private void editCategoryCommand(Number id) {
140     Optional<MenuCategoryViewModel> model = categoryTable.getItems()
141         .stream().filter(x -> x.getIdProperty().get() == (long) id)
142         .findFirst();
143
144     if (model.isPresent()) {
145         boolean result = ManageMenuCategoryDialog.show(getScene()
146             .getWindow(), model.get().toCategoryModel());
147         if (!result) {
148             viewModel.updateCategories();
149         }
150     }
151 }
152
153 private void deleteMenuCommand(Number x) {
154     viewModel.deleteMenuCommand(x);
155 }
156
157
```

```
158 private void editMenuCommand(Number id) {
159     Optional<MenuViewModel> model = menuTable.getItems().stream()
160         .filter(x -> x.getIdProperty().get() == (long) id).findFirst();
161
162     if (model.isPresent()) {
163         boolean result = ManageMenuDialog.show(getScene().getWindow(),
164             model.get().toMenuModel());
165         if (!result) {
166             viewModel.updateMenus();
167         }
168     }
169 }
170
171 private void addCategoryCommand() {
172     boolean result = ManageMenuCategoryDialog.show(getScene().getWindow(),
173         null);
174     if (!result) {
175         viewModel.updateCategories();
176     }
177 }
178
179 private void addMenuCommand() {
180     boolean result = ManageMenuDialog.show(getScene().getWindow(), null);
181     if (!result) {
182         viewModel.updateMenus();
183     }
184 }
185
186 public void show() {
187     viewModel.updateCategories();
188     viewModel.updateMenus();
189 }
190 }
```

gui/pages/ManageUsersPage.java

```
1 package at.lumetsnet.caas.gui.pages;
2
3 import java.util.Optional;
4
5 import javafx.scene.Node;
6 import javafx.scene.control.Button;
7 import javafx.scene.control.Label;
8 import javafx.scene.control.TableColumn;
9 import javafx.scene.control.TableView;
10 import javafx.scene.layout.HBox;
11 import javafx.scene.layout.Priority;
12 import javafx.scene.layout.VBox;
13 import at.lumetsnet.caas.gui.ManageUserActionCell;
```



```
14 import at.lumetsnet.caas.gui.dialogs.ManageUserDialog;
15 import at.lumetsnet.caas.viewmodel.ManageUsersPageViewModel;
16 import at.lumetsnet.caas.viewmodel.UserViewModel;
17
18 /**
19  * View page used for managing users Uses a ManageUsersPageViewModel for
20  * business logic operations
21  *
22  * @author romanlum
23  *
24  */
25 public class ManageUsersPage extends VBox implements Showable {
26
27     TableView<UserViewModel> table;
28     ManageUsersPageViewModel viewModel;
29
30     public ManageUsersPage() {
31
32         viewModel = new ManageUsersPageViewModel();
33         getStyleClass().add("page-content-container");
34
35         HBox topPane = new HBox();
36         topPane.getChildren().add(createTitle("Benutzerliste"));
37         HBox commandContainer = new HBox();
38         commandContainer.getStyleClass().add("page-command-container");
39         commandContainer.getChildren().add(createAddButton());
40         HBox.setHgrow(commandContainer, Priority.ALWAYS);
41
42         topPane.getChildren().add(commandContainer);
43         table = createTableView();
44         table.setItems(viewModel.getUserList());
45         getChildren().addAll(topPane, table);
46
47     }
48
49     private Node createAddButton() {
50         Button button = new Button("User anlegen");
51         button.getStyleClass().add("page-command");
52         button.setOnAction(x -> addUserCommand());
53         return button;
54     }
55
56     private Label createTitle(String title) {
57         Label titleLabel = new Label(title);
58         titleLabel.getStyleClass().add("page-title");
59         return titleLabel;
60     }
61
62     private TableView<UserViewModel> createTableView() {
```

```
63     TableView<UserViewModel> table = new TableView<>();
64     table.getStyleClass().add("table");
65     table.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
66
67     TableColumn<UserViewModel, String> column = new TableColumn<>(
68         "Benutzername");
69     column.setCellValueFactory(x -> x.getValue().getUserNameProperty());
70     table.getColumns().add(column);
71
72     column = new TableColumn<>("Vorname");
73     column.setCellValueFactory(x -> x.getValue().getFirstNameProperty());
74     table.getColumns().add(column);
75
76     column = new TableColumn<>("Nachname");
77     column.setCellValueFactory(x -> x.getValue().getLastNameProperty());
78     table.getColumns().add(column);
79
80     column = new TableColumn<>("Status");
81     column.setCellValueFactory(x -> x.getValue().getLockedStringProperty());
82     table.getColumns().add(column);
83
84     TableColumn<UserViewModel, Number> actionColumn = new TableColumn<>(
85         "Aktion");
86     actionColumn.setCellValueFactory(x -> x.getValue().getIdProperty());
87     actionColumn.setCellFactory(p -> new ManageUserActionCell<>(
88         x -> editCommand(x), x -> deleteCommand(x),
89         x -> toggleLockStateCommand(x)));
90
91     table.getColumns().add(actionColumn);
92     return table;
93 }
94
95 private void toggleLockStateCommand(Number x) {
96     viewModel.toggleLockStateCommand(x);
97 }
98
99 private void deleteCommand(Number x) {
100     viewModel.deleteCommand(x);
101 }
102 }
103
104 private void editCommand(Number id) {
105     Optional<UserViewModel> model = table.getItems().stream()
106         .filter(x -> x.getIdProperty().get() == (long) id).findFirst();
107
108     if (model.isPresent()) {
109         // call edit dialog
110         boolean result = ManageUserDialog.show(getScene().getWindow(),
111             model.get().toUserModel());
```

```
112     if (!result) {
113         viewModel.update();
114     }
115 }
116 }
117
118 private void addUserCommand() {
119     // call add dialog
120     boolean result = ManageUserDialog.show(getScene().getWindow(), null);
121     if (!result) {
122         viewModel.update();
123     }
124 }
125
126 public void show() {
127     viewModel.update();
128 }
129 }
```

gui/pages/OrdersPage.java

```
1 package at.lumetsnet.caas.gui.pages;
2
3 import javafx.scene.control.Label;
4 import javafx.scene.control.TableColumn;
5 import javafx.scene.control.TableColumn.SortType;
6 import javafx.scene.control.TableView;
7 import javafx.scene.layout.VBox;
8 import at.lumetsnet.caas.viewmodel.OrderViewModel;
9 import at.lumetsnet.caas.viewmodel.OrdersPageViewModel;
10
11 /**
12  * View page used for showing the orders Uses a OrdersPageViewModel for business
13  * logic operations
14  *
15  * @author romanlum
16  *
17  */
18 public class OrdersPage extends VBox implements Showable {
19
20     TableView<OrderViewModel> table;
21     OrdersPageViewModel viewModel;
22
23     public OrdersPage() {
24
25         viewModel = new OrdersPageViewModel();
26         getStyleClass().add("page-content-container");
27
28         Label title = createTitle("Heutige Bestellungen");
```

```
29     getChildren().add(title);
30
31     table = createTableView();
32     table.setItems(viewModel.getOrders());
33     getChildren().add(table);
34
35 }
36
37 private Label createTitle(String title) {
38     Label titleLabel = new Label(title);
39     titleLabel.getStyleClass().add("page-title");
40     return titleLabel;
41 }
42
43 private TableView<OrderViewModel> createTableView() {
44     TableView<OrderViewModel> table = new TableView<>();
45     table.getStyleClass().add("table");
46     table.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
47     TableColumn<OrderViewModel, String> column = new TableColumn<>(
48         "Benutzer");
49     column.setCellValueFactory(x -> x.getValue().getUserNameProperty());
50     table.getColumns().add(column);
51
52     column = new TableColumn<>("Gericht");
53     column.setCellValueFactory(x -> x.getValue().getMenuProperty());
54     table.getColumns().add(column);
55
56     column = new TableColumn<>("Zeit");
57     column.setCellValueFactory(x -> x.getValue().getTimeProperty());
58     column.setSortType(SortType.ASCENDING);
59     table.getColumns().add(column);
60     table.getSortOrder().clear();
61     table.getSortOrder().add(column);
62
63     column = new TableColumn<>("Kommentar");
64     column.setCellValueFactory(x -> x.getValue().getCommentProperty());
65
66     table.getColumns().add(column);
67
68     return table;
69 }
70
71 public void show() {
72     viewModel.update();
73     table.sort();
74 }
75 }
```

gui/pages/Showable.java

```
1 package at.lumetsnet.caas.gui.pages;
2
3 /**
4  * Simple interface used for marking pages as showable
5  *
6  * @author romanlum
7  *
8  */
9 public interface Showable {
10
11     public void show();
12 }
```

model/Entity.java

```
1 package at.lumetsnet.caas.model;
2
3 import java.io.Serializable;
4
5 /**
6  * Base data entity
7  *
8  * @author romanlum
9  *
10 */
11 public class Entity implements Serializable {
12     /**
13      *
14      */
15     private static final long serialVersionUID = 8439956639376975911L;
16     protected long id;
17
18     /**
19      * @return the id
20      */
21     public long getId() {
22         return id;
23     }
24
25     /**
26      * @param id
27      *         the id to set
28      */
29     public void setId(long id) {
30         this.id = id;
31     }
32 }
```

model/Menu.java

```
1 package at.lumetsnet.caas.model;
2
3 import java.time.LocalDate;
4
5 /**
6  * Menu data entity
7  *
8  * @author romanlum
9  *
10 */
11
12 public class Menu extends Entity {
13
14     /**
15      *
16      */
17     private static final long serialVersionUID = 2683509052388984064L;
18
19     private String description;
20     // price is stored in smallest currency unit!
21     private long price;
22     private LocalDate begin;
23     private LocalDate end;
24     private MenuCategory category;
25
26     public Menu() {
27     }
28
29     /**
30      * @param id
31      * @param description
32      * @param price
33      * @param begin
34      * @param end
35      * @param category
36      */
37     public Menu(long id, String description, long price, LocalDate begin,
38                 LocalDate end, MenuCategory category) {
39         this.id = id;
40         this.description = description;
41         this.price = price;
42         this.begin = begin;
43         this.end = end;
44         this.category = category;
45     }
46
47     /**
```

```
48     * @return the description
49     */
50     public String getDescription() {
51         return description;
52     }
53
54     /**
55     * @param description
56     *         the description to set
57     */
58     public void setDescription(String description) {
59         this.description = description;
60     }
61
62     /**
63     * @return the price in the smallest currency unit
64     */
65     public long getPrice() {
66         return price;
67     }
68
69     /**
70     * @param price
71     *         the price to set this price is set in the smallest currency
72     *         unit
73     */
74     public void setPrice(long price) {
75         this.price = price;
76     }
77
78     /**
79     * @return the begin
80     */
81     public LocalDate getBegin() {
82         return begin;
83     }
84
85     /**
86     * @param begin
87     *         the begin to set
88     */
89     public void setBegin(LocalDate begin) {
90         this.begin = begin;
91     }
92
93     /**
94     * @return the end
95     */
96     public void setEnd(LocalDate end) {
```

```
97 public LocalDate getEnd() {
98     return end;
99 }
100
101 /**
102  * @param end
103  *         the end to set
104  */
105 public void setEnd(LocalDate end) {
106     this.end = end;
107 }
108
109 /**
110  * @return the category
111  */
112 public MenuCategory getCategory() {
113     return category;
114 }
115
116 /**
117  * @param category
118  *         the category to set
119  */
120 public void setCategory(MenuCategory category) {
121     this.category = category;
122 }
123
124 /**
125  * @return the categoryId
126  */
127 public long getCategoryId() {
128     return category != null ? category.getId() : -1;
129 }
130
131 /**
132  * @param categoryId
133  *         the categoryId to set
134  */
135 public void setCategoryId(long categoryId) {
136     this.category = new MenuCategory();
137     this.category.setId(categoryId);
138 }
139
140 }
```

model/MenuCategory.java

```
1 package at.lumetsnet.caas.model;
2
```



```
3  /**
4   * MenuCategory data entity
5   *
6   * @author romanlum
7   *
8   */
9
10 public class MenuCategory extends Entity {
11
12     /**
13      *
14      */
15     private static final long serialVersionUID = -6364790602658825179L;
16     private String name;
17
18     public MenuCategory() {
19     }
20
21     /**
22      * @param id
23      * @param name
24      */
25     public MenuCategory(long id, String name) {
26         this.id = id;
27         this.name = name;
28     }
29
30     /**
31      * @return the name
32      */
33     public String getName() {
34         return name;
35     }
36
37     /**
38      * @param name
39      *         the name to set
40      */
41     public void setName(String name) {
42         this.name = name;
43     }
44 }
45 }
```

model/Order.java

```
1 package at.lumetsnet.caas.model;
2
3 import java.time.LocalDateTime;
```

```
4
5 /**
6  * Order data entity
7  *
8  * @author romanlum
9  *
10 */
11
12 public class Order extends Entity {
13
14     /**
15      *
16      */
17     private static final long serialVersionUID = -9188235335595584261L;
18     private Menu menu;
19     private User user;
20     private LocalDateTime time;
21     private String comment;
22
23     public Order() {
24     }
25
26     /**
27      * @param id
28      * @param menu
29      * @param user
30      * @param time
31      * @param comment
32      */
33     public Order(long id, Menu menu, User user, LocalDateTime time,
34                 String comment) {
35         this.id = id;
36         this.menu = menu;
37         this.user = user;
38         this.time = time;
39         this.comment = comment;
40     }
41
42     /**
43      * @return the menu
44      */
45     public Menu getMenu() {
46         return menu;
47     }
48
49     /**
50      * @param menu
51      *          the menu to set
52      */
53 }
```

```
53 public void setMenu(Menu menu) {
54     this.menu = menu;
55 }
56
57 /**
58  * @return the user
59  */
60 public User getUser() {
61     return user;
62 }
63
64 /**
65  * @param user
66  *         the user to set
67  */
68 public void setUser(User user) {
69     this.user = user;
70 }
71
72 /**
73  * @return the comment
74  */
75 public String getComment() {
76     return comment;
77 }
78
79 /**
80  * @param comment
81  *         the comment to set
82  */
83 public void setComment(String comment) {
84     this.comment = comment;
85 }
86
87 /**
88  * @return the time
89  */
90 public LocalDateTime getTime() {
91     return time;
92 }
93
94 /**
95  * @param time
96  *         the time to set
97  */
98 public void setTime(LocalDateTime time) {
99     this.time = time;
100 }
101
```

```
102  /**
103   * @return the menuId
104   */
105  public long getMenuId() {
106      return menu != null ? menu.getId() : -1;
107  }
108
109  /**
110   * @param menuId
111   *         the menuId to set
112   */
113  public void setMenuId(long menuId) {
114      this.menu = new Menu();
115      this.menu.setId(menuId);
116  }
117
118  /**
119   * @return the userId
120   */
121  public long getUserId() {
122      return user != null ? user.getId() : -1;
123  }
124
125  /**
126   * @param userId
127   *         the userId to set
128   */
129  public void setUserId(long userId) {
130      this.user = new User();
131      this.user.setId(userId);
132  }
133
134 }
```

model/User.java

```
1 package at.lumetsnet.caas.model;
2
3 import java.io.Serializable;
4
5 /**
6  * user data entity
7  *
8  * @author romanlum
9  *
10 */
11
12 public class User extends Entity implements Serializable {
13
```

```
14  /**
15   *
16   */
17  private static final long serialVersionUID = -9121752719148285484L;
18
19  private String userName;
20  private String password;
21  private String firstName;
22  private String lastName;
23  private boolean locked;
24
25  public User() {
26  }
27
28  /**
29   * @param id
30   * @param userName
31   * @param password
32   * @param firstName
33   * @param lastName
34   * @param locked
35   * @param isAdmin
36   */
37  public User(long id, String userName, String password, String firstName,
38             String lastName, boolean locked) {
39      this.id = id;
40      this.userName = userName;
41      this.password = password;
42      this.firstName = firstName;
43      this.lastName = lastName;
44      this.locked = locked;
45  }
46
47  /**
48   * @return the userName
49   */
50  public String getUserName() {
51      return userName;
52  }
53
54  /**
55   * @param userName
56   *      the userName to set
57   */
58  public void setUserName(String userName) {
59      this.userName = userName;
60  }
61
62  /**
```

```
63     * @return the password
64     */
65     public String getPassword() {
66         return password;
67     }
68
69     /**
70     * @param password
71     *         the password to set
72     */
73     public void setPassword(String password) {
74         this.password = password;
75     }
76
77     /**
78     * @return the firstName
79     */
80     public String getFirstName() {
81         return firstName;
82     }
83
84     /**
85     * @param firstName
86     *         the firstName to set
87     */
88     public void setFirstName(String firstName) {
89         this.firstName = firstName;
90     }
91
92     /**
93     * @return the lastName
94     */
95     public String getLastName() {
96         return lastName;
97     }
98
99     /**
100    * @param lastName
101    *         the lastName to set
102    */
103    public void setLastName(String lastName) {
104        this.lastName = lastName;
105    }
106
107    /**
108    * @return the locked
109    */
110    public boolean isLocked() {
111        return locked;
```

```
112     }
113
114     /**
115      * @param locked
116      *         the locked to set
117      */
118     public void setLocked(boolean locked) {
119         this.locked = locked;
120     }
121
122 }
```

rmi/RemoteServiceMain.java

```
1 package at.lumetsnet.caas.rmi;
2
3 import java.net.MalformedURLException;
4 import java.rmi.Naming;
5 import java.rmi.Remote;
6 import java.rmi.RemoteException;
7 import java.rmi.registry.LocateRegistry;
8 import java.rmi.server.UnicastRemoteObject;
9
10 import at.lumetsnet.caas.rmi.impl.MenuServiceImpl;
11 import at.lumetsnet.caas.rmi.impl.OrderServiceImpl;
12 import at.lumetsnet.caas.rmi.impl.UserServiceImpl;
13
14 /**
15  * Rmi remote service class
16  *
17  * @author romanlum
18  */
19 */
20 public class RemoteServiceMain {
21
22     public static void main(String[] args) {
23         try {
24             String host = "localhost:1099";
25             if (args.length == 1) {
26                 host = args[0];
27             }
28
29             LocateRegistry.createRegistry(getPort(host));
30             registerService(new UserServiceImpl(), host, "CaasUserService");
31             registerService(new MenuServiceImpl(), host, "CaasMenuService");
32             registerService(new OrderServiceImpl(), host, "CaasOrderService");
33
34         } catch (Exception ex) {
35             System.out.println("Could not register services");
36         }
37     }
38 }
```

```
36     ex.printStackTrace();
37 }
38
39 }
40
41 /**
42  * Creates and exports the service
43  *
44  * @param remote
45  * @param host
46  * @param name
47  */
48 public static void registerSerice(Remote remote, String host, String name) {
49     Remote serviceStub;
50     try {
51         serviceStub = UnicastRemoteObject.exportObject(remote, 0);
52         Naming.rebind("rmi://" + host + "/" + name, serviceStub);
53         System.out.println(name + " available on port " + "rmi://" + host
54             + "/" + name);
55     } catch (RemoteException | MalformedURLException e) {
56         System.out.println("Could not register service " + name);
57         e.printStackTrace();
58     }
59 }
60
61
62 /**
63  * Gets the port from host:port string Defaults to 1099 (RMI default port)
64  *
65  * @param hostPort
66  * @return
67  */
68 private static int getPort(String hostPort) {
69     int idx = hostPort.lastIndexOf(':');
70     if (idx == -1) {
71         return 1099;
72     } else {
73         return Integer.parseInt(hostPort.substring(idx + 1));
74     }
75 }
76
77 }
```

rmi/impl/MenuServiceImpl.java

```
1 package at.lumetsnet.caas.rmi.impl;
2
3 import java.util.Collection;
4
```



```
5 import at.lumetsnet.caas.dal.MenuCategoryDao;
6 import at.lumetsnet.caas.dal.MenuCategoryDaoJdbc;
7 import at.lumetsnet.caas.dal.MenuDao;
8 import at.lumetsnet.caas.dal.MenuDaoJdbc;
9 import at.lumetsnet.caas.model.Menu;
10 import at.lumetsnet.caas.model.MenuCategory;
11 import at.lumetsnet.caas.rmi.interfaces.RemoteMenuService;
12
13 /**
14  * MenuService impl using jdbc
15  *
16  * @author romanlum
17  *
18  */
19 public class MenuServiceImpl extends ServiceImpl implements RemoteMenuService {
20
21     private MenuDao menuDao;
22     private MenuCategoryDao categoryDao;
23
24     public MenuServiceImpl() {
25         menuDao = new MenuDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
26         categoryDao = new MenuCategoryDaoJdbc(CONNECTION_STRING, USER_NAME,
27             PASSWORD);
28     }
29
30     /**
31      * Fetches all categories
32      *
33      * @return
34      */
35     public Collection<MenuCategory> getAllCategories() {
36         return categoryDao.getAll();
37     }
38
39     /**
40      * Deletes the category with the given id
41      *
42      * @param id
43      */
44     public void deleteCategory(long id) {
45         categoryDao.delete(id);
46     }
47
48     /**
49      * Deletes the menu with the given id
50      *
51      * @param id
52      */
53     public void deleteMenu(long id) {
```

```
54     menuDao.delete(id);
55 }
56
57 /**
58  * Saves or adds the category
59  *
60  * @param data
61  */
62 public void saveOrUpdateCategory(MenuCategory data) {
63     categoryDao.saveOrUpdate(data);
64 }
65
66 /**
67  * Saves or adds the menu
68  *
69  * @param data
70  */
71 public void saveOrUpdateMenu(Menu data) {
72     menuDao.saveOrUpdate(data);
73 }
74
75 /**
76  * Fetches all menus
77  *
78  * @return
79  */
80 public Collection<Menu> getAllMenus() {
81     Collection<Menu> data = menuDao.getAll();
82     data.stream().forEach(x -> {
83         x.setCategory(categoryDao.get(x.getCategoryId()));
84     });
85     return data;
86 }
87 }
88 }
```

rmi/impl/OrderServiceImpl.java

```
1 package at.lumetsnet.caas.rmi.impl;
2
3 import java.rmi.RemoteException;
4 import java.time.LocalDate;
5 import java.util.Collection;
6
7 import at.lumetsnet.caas.dal.MenuDao;
8 import at.lumetsnet.caas.dal.MenuDaoJdbc;
9 import at.lumetsnet.caas.dal.OrderDao;
10 import at.lumetsnet.caas.dal.OrderDaoJdbc;
11 import at.lumetsnet.caas.dal.UserDao;
```

```
12 import at.lumetsnet.caas.dal.UserDaoJdbc;
13 import at.lumetsnet.caas.model.Order;
14 import at.lumetsnet.caas.rmi.interfaces.RemoteOrderService;
15
16 /**
17  * Order service impl using jdbc
18  *
19  * @author romanlum
20  *
21  */
22 public class OrderServiceImpl extends ServiceImpl implements RemoteOrderService {
23     private OrderDao orderDao;
24     private MenuDao menuDao;
25     private UserDao userDao;
26
27     public OrderServiceImpl() {
28         orderDao = new OrderDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
29         menuDao = new MenuDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
30         userDao = new UserDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
31     }
32
33     @Override
34     public Collection<Order> getTodaysOrders() throws RemoteException {
35         Collection<Order> result = orderDao.getOrdersByDate(LocalDate.now());
36         // fetch data for referenced objects
37         result.stream().forEach(x -> {
38             x.setMenu(menuDao.get(x.getMenuId()));
39             x.setUser(userDao.get(x.getUserId()));
40         });
41         return result;
42     }
43
44 }
```

rmi/impl/ServiceImpl.java

```
1 package at.lumetsnet.caas.rmi.impl;
2
3 /**
4  * Base class for all service impls
5  *
6  * @author romanlum
7  *
8  */
9 public class ServiceImpl {
10
11     protected static final String CONNECTION_STRING = "jdbc:mysql://localhost/CaasDb";
12     protected static final String USER_NAME = "root";
13     protected static final String PASSWORD = null;
```

14
15 }

rmi/impl/UserServiceImpl.java

```
1 package at.lumetsnet.caas.rmi.impl;
2
3 import java.rmi.RemoteException;
4 import java.util.Collection;
5
6 import at.lumetsnet.caas.dal.UserDao;
7 import at.lumetsnet.caas.dal.UserDaoJdbc;
8 import at.lumetsnet.caas.model.User;
9 import at.lumetsnet.caas.rmi.interfaces.RemoteUserService;
10
11 /**
12  * User service impl using jdbc
13  *
14  * @author romanlum
15  *
16  */
17 public class UserServiceImpl extends ServiceImpl implements RemoteUserService {
18
19     private UserDao dao;
20
21     public UserServiceImpl() {
22         dao = new UserDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
23     }
24
25     /**
26      * Fetches all users
27      *
28      * @return
29      */
30     public Collection<User> getAllUsers() throws RemoteException {
31         return dao.getAll();
32     }
33
34     /**
35      * delete the user
36      *
37      * @param id
38      */
39     public void deleteUser(long id) throws RemoteException {
40         dao.delete(id);
41     }
42
43     /**
44      * Toggles the locked state of a user
```

```
45  *
46  * @param id
47  */
48  public void toggleLockState(long id) throws RemoteException {
49      User user = dao.get(id);
50      if (user != null) {
51          user.setLocked(!user.isLocked());
52      }
53      saveOrUpdate(user);
54  }
55
56  /**
57   * Saves or adds the user
58   *
59   * @param userModel
60   */
61  public void saveOrUpdate(User userModel) throws RemoteException {
62      dao.saveOrUpdate(userModel);
63  }
64
65 }
```

rmi/interfaces/RemoteMenuService.java

```
1  package at.lumetsnet.caas.rmi.interfaces;
2
3  import java.rmi.Remote;
4  import java.rmi.RemoteException;
5  import java.util.Collection;
6
7  import at.lumetsnet.caas.model.Menu;
8  import at.lumetsnet.caas.model.MenuCategory;
9
10 public interface RemoteMenuService extends Remote {
11
12     /**
13      * Fetches all categories
14      *
15      * @return
16      */
17     public Collection<MenuCategory> getAllCategories() throws RemoteException;
18
19     /**
20      * Deletes the category with the given id
21      *
22      * @param id
23      */
24     public void deleteCategory(long id) throws RemoteException;
25 }
```

```
26  /**
27   * Deletes the menu with the given id
28   *
29   * @param id
30   */
31  public void deleteMenu(long id) throws RemoteException;
32
33  /**
34   * Saves or adds the category
35   *
36   * @param data
37   */
38  public void saveOrUpdateCategory(MenuCategory data) throws RemoteException;
39
40  /**
41   * Saves or adds the menu
42   *
43   * @param data
44   */
45  public void saveOrUpdateMenu(Menu data) throws RemoteException;
46
47  /**
48   * Fetches all menus
49   *
50   * @return
51   */
52  public Collection<Menu> getAllMenus() throws RemoteException;
53 }
```

rmi/interfaces/RemoteOrderService.java

```
1  package at.lumetsnet.caas.rmi.interfaces;
2
3  import java.rmi.Remote;
4  import java.rmi.RemoteException;
5  import java.util.Collection;
6
7  import at.lumetsnet.caas.model.Order;
8
9  public interface RemoteOrderService extends Remote {
10
11     /**
12      * Fetches the orders of today
13      *
14      * @return
15      */
16     public Collection<Order> getTodaysOrders() throws RemoteException;
17
18 }
```

rmi/interfaces/RemoteUserService.java

```
1 package at.lumetsnet.caas.rmi.interfaces;
2
3 import java.rmi.Remote;
4 import java.rmi.RemoteException;
5 import java.util.Collection;
6
7 import at.lumetsnet.caas.model.User;
8
9 public interface RemoteUserService extends Remote {
10
11     /**
12      * Fetches all users
13      *
14      * @return
15      */
16     public Collection<User> getAllUsers() throws RemoteException;
17
18     /**
19      * delete the user
20      *
21      * @param id
22      */
23     public void deleteUser(long id) throws RemoteException;
24
25     /**
26      * Toggles the locked state of a user
27      *
28      * @param id
29      */
30     public void toggleLockState(long id) throws RemoteException;
31
32     /**
33      * Saves or adds the user
34      *
35      * @param userModel
36      */
37     public void saveOrUpdate(User userModel) throws RemoteException;
38 }
```

viewmodel/ManageMenusPageViewModel.java

```
1 package at.lumetsnet.caas.viewmodel;
2
3 import java.util.Collection;
```

```
4
5 import javafx.collections.FXCollections;
6 import javafx.collections.ObservableList;
7 import at.lumetsnet.caas.business.MenuService;
8 import at.lumetsnet.caas.model.Menu;
9 import at.lumetsnet.caas.model.MenuCategory;
10
11 /**
12  * ViewModel (logic) class for ManageMenusPage
13  *
14  * @author romanlum
15  *
16  */
17 public class ManageMenusPageViewModel {
18
19     private ObservableList<MenuCategoryViewModel> categories;
20     private ObservableList<MenuViewModel> menus;
21
22     public ManageMenusPageViewModel() {
23         categories = FXCollections.observableArrayList();
24         menus = FXCollections.observableArrayList();
25     }
26
27     public void updateCategories() {
28         Collection<MenuCategory> data = MenuService.getInstance()
29             .getAllCategories();
30         categories.clear();
31         if (data != null) {
32             data.forEach(x -> categories.add(new MenuCategoryViewModel(x)));
33         }
34     }
35
36     public void updateMenus() {
37         Collection<Menu> data = MenuService.getInstance().getAllMenus();
38         menus.clear();
39         if (data != null) {
40             data.forEach(x -> menus.add(new MenuViewModel(x)));
41         }
42     }
43
44     public ObservableList<MenuCategoryViewModel> getCategoryList() {
45         return categories;
46     }
47
48     public ObservableList<MenuViewModel> getMenuList() {
49         return menus;
50     }
51
52     public void deleteCategoryCommand(Number userId) {
```



```
53     MenuService.getInstance().deleteCategory((long) userId);
54     updateCategories();
55     updateMenus();
56
57 }
58
59 public void deleteMenuCommand(Number id) {
60     MenuService.getInstance().deleteMenu((long) id);
61     updateMenus();
62
63 }
64
65 }
```

viewmodel/ManageUsersPageViewModel.java

```
1 package at.lumetsnet.caas.viewmodel;
2
3 import java.util.Collection;
4
5 import javafx.collections.FXCollections;
6 import javafx.collections.ObservableList;
7 import at.lumetsnet.caas.business.UserService;
8 import at.lumetsnet.caas.model.User;
9
10 /**
11  * ViewModel (logic) class for ManageUsersPage
12  *
13  * @author romanlum
14  *
15  */
16
17 public class ManageUsersPageViewModel {
18
19     private ObservableList<UserViewModel> users;
20
21     public ManageUsersPageViewModel() {
22         users = FXCollections.observableArrayList();
23     }
24
25     public void update() {
26         Collection<User> data = UserService.getInstance().getAllUsers();
27         users.clear();
28         if (data != null) {
29             data.forEach(x -> users.add(new UserViewModel(x)));
30         }
31     }
32
33     public ObservableList<UserViewModel> getUserList() {
```

```
34     return users;
35 }
36
37 public void deleteCommand(Number userId) {
38     UserService.getInstance().deleteUser((long) userId);
39     update();
40 }
41
42
43 public void toggleLockStateCommand(Number userId) {
44     UserService.getInstance().toggleLockState((long) userId);
45     update();
46 }
47 }
```

viewmodel/MenuCategoryViewModel.java

```
1 package at.lumetsnet.caas.viewmodel;
2
3 import javafx.beans.property.LongProperty;
4 import javafx.beans.property.SimpleLongProperty;
5 import javafx.beans.property.SimpleStringProperty;
6 import javafx.beans.property.StringProperty;
7 import at.lumetsnet.caas.model.MenuCategory;
8
9 /**
10  * ViewModel wrapper for MenuCategory entity Uses javafx properties for
11  * databinding
12  *
13  * @author romanlum
14  *
15  */
16 public class MenuCategoryViewModel implements Validatable {
17
18     private LongProperty idProperty;
19     private StringProperty nameProperty;
20
21     public MenuCategoryViewModel(MenuCategory input) {
22         if (input == null) {
23             input = new MenuCategory();
24             input.setId(-1);
25         }
26         idProperty = new SimpleLongProperty(input.getId());
27         nameProperty = new SimpleStringProperty(input.getName());
28     }
29
30
31     public MenuCategory toCategoryModel() {
32         MenuCategory result = new MenuCategory();
```

```
33     result.setId(idProperty.get());
34     result.setName(nameProperty.get());
35     return result;
36
37 }
38
39 public boolean validate() {
40     return nameProperty.get() != null && !nameProperty.get().isEmpty();
41 }
42
43 @Override
44 public String toString() {
45     return getNameProperty().get();
46 }
47
48 /**
49  * @return the id
50  */
51 public LongProperty getIdProperty() {
52     return idProperty;
53 }
54
55 /**
56  * @return the userNameProperty
57  */
58 public StringProperty getNameProperty() {
59     return nameProperty;
60 }
61
62 }
```

viewmodel/MenuViewModel.java

```
1 package at.lumetsnet.caas.viewmodel;
2
3 import java.time.LocalDate;
4 import java.time.format.DateTimeFormatter;
5
6 import javafx.beans.property.LongProperty;
7 import javafx.beans.property.ObjectProperty;
8 import javafx.beans.property.SimpleLongProperty;
9 import javafx.beans.property.SimpleObjectProperty;
10 import javafx.beans.property.SimpleStringProperty;
11 import javafx.beans.property.StringProperty;
12 import at.lumetsnet.caas.model.Menu;
13
14 /**
15  * ViewModel wrapper for Menu entity Uses javafx properties for databinding
16  */
```

```
17  * @author romanlum
18  *
19  */
20
21 public class MenuViewModel implements Validatable {
22
23     private LongProperty idProperty;
24     private StringProperty descriptionProperty;
25     private LongProperty priceProperty;
26     private ObjectProperty<LocalDate> beginProperty;
27     private ObjectProperty<LocalDate> endProperty;
28     private ObjectProperty<MenuCategoryViewModel> categoryProperty;
29     private StringProperty usageRangeProperty;
30     private StringProperty priceAsStringProperty;
31
32     public MenuViewModel(Menu data) {
33         if (data == null) {
34             data = new Menu();
35             data.setId(-1);
36         }
37         idProperty = new SimpleLongProperty(data.getId());
38         descriptionProperty = new SimpleStringProperty(data.getDescription());
39         priceProperty = new SimpleLongProperty(data.getPrice());
40         beginProperty = new SimpleObjectProperty<LocalDate>(data.getBegin());
41         endProperty = new SimpleObjectProperty<LocalDate>(data.getEnd());
42         categoryProperty = new SimpleObjectProperty<>(
43             new MenuCategoryViewModel(data.getCategory()));
44         if (data.getBegin() != null && data.getEnd() != null) {
45             usageRangeProperty = new SimpleStringProperty(data.getBegin()
46                 .format(DateTimeFormatter.ISO_LOCAL_DATE)
47                 + " - "
48                 + data.getEnd().format(DateTimeFormatter.ISO_LOCAL_DATE));
49         } else {
50             usageRangeProperty = new SimpleStringProperty("");
51         }
52         priceAsStringProperty = new SimpleStringProperty(""
53             + (data.getPrice() / (double) 100));
54     }
55
56     public Menu toMenuModel() {
57         Menu menu = new Menu();
58         menu.setId(idProperty.get());
59         menu.setBegin(beginProperty.get());
60         menu.setEnd(endProperty.get());
61         menu.setDescription(descriptionProperty.get());
62
63         long amount = (long) (Double.parseDouble(priceAsStringProperty.get()) * 100);
64         menu.setPrice(amount);
65         menu.setCategory(categoryProperty.get().toCategoryModel());
66     }
67 }
```

```
66     return menu;
67
68 }
69
70 /**
71  * @return the idProperty
72  */
73 public LongProperty getIdProperty() {
74     return idProperty;
75 }
76
77 /**
78  * @return the descriptionNameProperty
79  */
80 public StringProperty getDescriptionProperty() {
81     return descriptionProperty;
82 }
83
84 /**
85  * @return the priceProperty
86  */
87 public LongProperty getPriceProperty() {
88     return priceProperty;
89 }
90
91 /**
92  * @return the beginProperty
93  */
94 public ObjectProperty<LocalDate> getBeginProperty() {
95     return beginProperty;
96 }
97
98 /**
99  * @return the endProperty
100  */
101 public ObjectProperty<LocalDate> getEndProperty() {
102     return endProperty;
103 }
104
105 /**
106  * @return the categoryProperty
107  */
108 public ObjectProperty<MenuCategoryViewModel> getCategoryProperty() {
109     return categoryProperty;
110 }
111
112 /**
113  * @return the usageRange
114  */
```

```
115 public StringProperty getUsageRangeProperty() {
116     return usageRangeProperty;
117 }
118
119 /**
120  * @return the priceAsStringProperty
121  */
122 public StringProperty getPriceAsStringProperty() {
123     return priceAsStringProperty;
124 }
125
126 @Override
127 public boolean validate() {
128     boolean val = descriptionProperty.get() != null
129         && !descriptionProperty.get().isEmpty()
130         && categoryProperty.get() != null
131         && categoryProperty.get().getIdProperty().get() != -1;
132
133     if (!val)
134         return val;
135     // extended validation of price
136     try {
137         Double.parseDouble(priceAsStringProperty.get());
138     } catch (NumberFormatException ex) {
139         return false;
140     }
141     return true;
142 }
143 }
144
145 }
```

viewmodel/OrdersPageViewModel.java

```
1 package at.lumetsnet.caas.viewmodel;
2
3 import java.util.Collection;
4
5 import javafx.collections.FXCollections;
6 import javafx.collections.ObservableList;
7 import at.lumetsnet.caas.business.OrderService;
8 import at.lumetsnet.caas.model.Order;
9
10 /**
11  * ViewModel (logic) class for OrdersPage
12  *
13  * @author romanlum
14  *
15  */
```

```
16 public class OrdersPageViewModel {
17
18     private ObservableList<OrderViewModel> orders;
19
20     public OrdersPageViewModel() {
21         orders = FXCollections.observableArrayList();
22     }
23
24     public void update() {
25         Collection<Order> orderData = OrderService.getInstance()
26             .getTodaysOrders();
27         orders.clear();
28         if (orderData != null) {
29             orderData.forEach(x -> orders.add(new OrderViewModel(x)));
30         }
31     }
32
33     public ObservableList<OrderViewModel> getOrders() {
34         return orders;
35     }
36 }
```

viewmodel/OrderViewModel.java

```
1 package at.lumetsnet.caas.viewmodel;
2
3 import java.time.format.DateTimeFormatter;
4
5 import javafx.beans.property.SimpleStringProperty;
6 import javafx.beans.property.StringProperty;
7 import at.lumetsnet.caas.model.Order;
8
9 /**
10  * ViewModel wrapper for order entity Uses javafx properties for databinding
11  *
12  * @author romanlum
13  *
14  */
15 public class OrderViewModel {
16
17     private StringProperty userNameProperty;
18     private StringProperty menuProperty;
19     private StringProperty timeProperty;
20     private StringProperty commentProperty;
21
22     public OrderViewModel(Order order) {
23         userNameProperty = new SimpleStringProperty(order.getUser()
24             .getFirstName() + " " + order.getUser().getLastName());
25         menuProperty = new SimpleStringProperty(order.getMenu()
```

```
26         .getDescription());
27     timeProperty = new SimpleStringProperty(order.getTime().format(
28         DateTimeFormatter.ofPattern("HH:mm")));
29     commentProperty = new SimpleStringProperty(order.getComment());
30 }
31
32 /**
33  * @return the userNameProperty
34  */
35 public StringProperty getUserNameProperty() {
36     return userNameProperty;
37 }
38
39 /**
40  * @return the menuProperty
41  */
42 public StringProperty getMenuProperty() {
43     return menuProperty;
44 }
45
46 /**
47  * @return the timeProperty
48  */
49 public StringProperty getTimeProperty() {
50     return timeProperty;
51 }
52
53 /**
54  * @return the commentProperty
55  */
56 public StringProperty getCommentProperty() {
57     return commentProperty;
58 }
59
60 }
```

viewmodel/UserViewModel.java

```
1 package at.lumetsnet.caas.viewmodel;
2
3 import javafx.beans.property.BooleanProperty;
4 import javafx.beans.property.LongProperty;
5 import javafx.beans.property.SimpleBooleanProperty;
6 import javafx.beans.property.SimpleLongProperty;
7 import javafx.beans.property.SimpleStringProperty;
8 import javafx.beans.property.StringProperty;
9 import at.lumetsnet.caas.model.User;
10
11 /**
```



```
12  * ViewModel wrapper for User entity Uses javafx properties for databinding
13  *
14  * @author romanlum
15  *
16  */
17
18 public class UserViewModel implements Validatable {
19
20     private LongProperty idProperty;
21     private StringProperty userNameProperty;
22     private StringProperty passwordProperty;
23     private StringProperty firstNameProperty;
24     private StringProperty lastNameProperty;
25     private BooleanProperty lockedProperty;
26
27     public UserViewModel(User user) {
28         if (user == null) {
29             user = new User();
30             user.setId(-1);
31         }
32         idProperty = new SimpleLongProperty(user.getId());
33         userNameProperty = new SimpleStringProperty(user.getUserName());
34         passwordProperty = new SimpleStringProperty(user.getPassword());
35         firstNameProperty = new SimpleStringProperty(user.getFirstName());
36         lastNameProperty = new SimpleStringProperty(user.getLastName());
37         lockedProperty = new SimpleBooleanProperty(user.isLocked());
38
39     }
40
41     public User toUserModel() {
42         User result = new User();
43         result.setId(idProperty.get());
44         result.setUserName(userNameProperty.get());
45         result.setPassword(passwordProperty.get());
46         result.setFirstName(firstNameProperty.get());
47         result.setLastName(lastNameProperty.get());
48         result.setLocked(lockedProperty.get());
49         return result;
50     }
51 }
52
53 /**
54  * @return the id
55  */
56 public LongProperty getIdProperty() {
57     return idProperty;
58 }
59
60 /**
```

```
61     * @return the userNameProperty
62     */
63     public StringProperty getUsernameProperty() {
64         return userNameProperty;
65     }
66
67     /**
68     * @return the passwordProperty
69     */
70     public StringProperty getPasswordProperty() {
71         return passwordProperty;
72     }
73
74     /**
75     * @return the firstNameProperty
76     */
77     public StringProperty getFirstNameProperty() {
78         return firstNameProperty;
79     }
80
81     /**
82     * @return the lastNameProperty
83     */
84     public StringProperty getLastNameProperty() {
85         return lastNameProperty;
86     }
87
88     /**
89     * @return the lockedProperty
90     */
91     public BooleanProperty getLockedProperty() {
92         return lockedProperty;
93     }
94
95     /**
96     * @return the lockedProperty as string
97     */
98     public StringProperty getLockedStringProperty() {
99         return new SimpleStringProperty(lockedProperty.get() ? "Gesperrt"
100             : "Ok");
101     }
102
103     @Override
104     public boolean validate() {
105         return userNameProperty.get() != null
106             && !userNameProperty.get().isEmpty()
107             && passwordProperty.get() != null
108             && !passwordProperty.get().isEmpty()
109             && firstNameProperty.get() != null
```

```
110         && !firstNameProperty.get().isEmpty()
111         && lastNameProperty.get() != null
112         && !lastNameProperty.get().isEmpty();
113
114     }
115
116 }
```

viewmodel/Validatable.java

```
1 package at.lumetsnet.caas.viewmodel;
2
3 /**
4  * Interface used for validating entity view models
5  *
6  * @author romanlum
7  *
8  */
9 public interface Validatable {
10
11     /**
12      * validates the entity
13      *
14      * @return true on successfull validation otherwise false
15      */
16     boolean validate();
17 }
```

1.4 Test - Sourcecode - Java

GenericDaoTest.java

```
1 package at.lumetsnet.caas.test;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.PreparedStatement;
6 import java.sql.SQLException;
7
8 import at.lumetsnet.caas.dal.DataAccessException;
9
10 public class GenericDaoTest {
11
12     protected static final String CONNECTION_STRING = "jdbc:mysql://localhost/CaasDb";
13     protected static final String USER_NAME = "root";
14     protected static final String PASSWORD = null;
15
16     public void cleanTable(String table) {
17         try (Connection con = DriverManager.getConnection(CONNECTION_STRING,
18             USER_NAME, PASSWORD)) {
19             // clean table
20             try (PreparedStatement stmt = con.prepareStatement(String.format(
21                 "Delete from '%s'", table))) {
22                 stmt.executeUpdate();
23             }
24         } catch (SQLException ex) {
25             throw new DataAccessException(ex.getMessage());
26         }
27     }
28 }
```

MenuCategoryDaoTest.java

```
1 package at.lumetsnet.caas.test;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.assertTrue;
5
6 import org.junit.Before;
7 import org.junit.Test;
8
9 import at.lumetsnet.caas.dal.MenuCategoryDaoJdbc;
10 import at.lumetsnet.caas.model.MenuCategory;
11
12 public class MenuCategoryDaoTest extends GenericDaoTest {
13 }
```

```
14 private MenuCategoryDaoJdbc dao;
15
16 @Before
17 public void setUp() {
18     cleanTable("Order");
19     cleanTable("Menu");
20     cleanTable("MenuCategory");
21     cleanTable("User");
22
23     dao = new MenuCategoryDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
24 }
25
26 @Test
27 public void insertTest() {
28
29     MenuCategory entity = new MenuCategory(-1, "category");
30     dao.saveOrUpdate(entity);
31     assertTrue(entity.getId() != -1);
32 }
33
34 @Test
35 public void updateTest() {
36     MenuCategory entity = new MenuCategory(-1, "category");
37     dao.saveOrUpdate(entity);
38     entity.setName("newcat");
39     dao.saveOrUpdate(entity);
40     MenuCategory dbEntry = dao.get(entity.getId());
41     assertEquals("newcat", dbEntry.getName());
42 }
43
44 @Test
45 public void getTest() {
46     MenuCategory entity = new MenuCategory(-1, "category");
47     dao.saveOrUpdate(entity);
48
49     MenuCategory dbEntry = dao.get(entity.getId());
50     assertEquals(entity.getName(), dbEntry.getName());
51
52 }
53
54 @Test
55 public void getAllTest() {
56     MenuCategory entity = new MenuCategory(-1, "category");
57     dao.saveOrUpdate(entity);
58     entity = new MenuCategory(-1, "category II");
59     dao.saveOrUpdate(entity);
60     assertEquals(2, dao.getAll().size());
61 }
62
```

```
63  @Test
64  public void deleteTest() {
65      MenuCategory entity = new MenuCategory(-1, "category");
66      dao.saveOrUpdate(entity);
67      assertTrue(dao.get(entity.getId()) != null);
68      dao.delete(entity.getId());
69      assertTrue(dao.get(entity.getId()) == null);
70  }
71
72 }
```

MenuDaoTest.java

```
1  package at.lumetsnet.caas.test;
2
3  import static org.junit.Assert.assertEquals;
4  import static org.junit.Assert.assertTrue;
5
6  import java.time.LocalDate;
7
8  import org.junit.Before;
9  import org.junit.Test;
10
11  import at.lumetsnet.caas.dal.MenuCategoryDao;
12  import at.lumetsnet.caas.dal.MenuCategoryDaoJdbc;
13  import at.lumetsnet.caas.dal.MenuDao;
14  import at.lumetsnet.caas.dal.MenuDaoJdbc;
15  import at.lumetsnet.caas.model.Menu;
16  import at.lumetsnet.caas.model.MenuCategory;
17
18  public class MenuDaoTest extends GenericDaoTest {
19
20      private MenuDao dao;
21      private MenuCategoryDao categoryDao;
22
23      private long catId;
24
25      @Before
26      public void setUp() {
27          cleanTable("Order");
28          cleanTable("Menu");
29          cleanTable("MenuCategory");
30          cleanTable("User");
31
32          dao = new MenuDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
33          categoryDao = new MenuCategoryDaoJdbc(CONNECTION_STRING, USER_NAME,
34              PASSWORD);
35          MenuCategory cat = new MenuCategory(-1, "Cat1");
36          categoryDao.saveOrUpdate(cat);
```

```
37     catId = cat.getId();
38
39 }
40
41 @Test
42 public void insertTest() {
43
44     Menu entity = new Menu(-1, "desc", 1000, LocalDate.now(), LocalDate
45         .now().plusDays(100), null);
46     entity.setCategoryId(catId);
47     dao.saveOrUpdate(entity);
48     assertTrue(entity.getId() != -1);
49 }
50
51 @Test
52 public void updateTest() {
53     Menu entity = new Menu(-1, "desc", 1000, LocalDate.now(), LocalDate
54         .now().plusDays(100), null);
55     entity.setCategoryId(catId);
56     dao.saveOrUpdate(entity);
57     entity.setDescription("new desc");
58     dao.saveOrUpdate(entity);
59     Menu dbEntry = dao.get(entity.getId());
60     assertEquals("new desc", dbEntry.getDescription());
61 }
62
63 @Test
64 public void getTest() {
65     Menu entity = new Menu(-1, "desc", 1000, LocalDate.now(), LocalDate
66         .now().plusDays(100), null);
67     entity.setCategoryId(catId);
68     dao.saveOrUpdate(entity);
69
70     Menu dbEntity = dao.get(entity.getId());
71     assertEquals(entity.getBegin(), dbEntity.getBegin());
72     assertEquals(entity.getCategoryId(), dbEntity.getCategoryId());
73     assertEquals(entity.getDescription(), dbEntity.getDescription());
74     assertEquals(entity.getEnd(), dbEntity.getEnd());
75     assertEquals(entity.getPrice(), dbEntity.getPrice());
76
77 }
78
79 @Test
80 public void getAllTest() {
81     Menu entity = new Menu(-1, "desc", 1000, LocalDate.now(), LocalDate
82         .now().plusDays(100), null);
83     entity.setCategoryId(catId);
84     dao.saveOrUpdate(entity);
85     entity = new Menu(-1, "desc2", 2000, LocalDate.now(), LocalDate.now()
```

```
86         .plusDays(100), null);
87     entity.setCategoryId(catId);
88
89     dao.saveOrUpdate(entity);
90     assertEquals(2, dao.getAll().size());
91 }
92
93 @Test
94 public void deleteTest() {
95     Menu entity = new Menu(-1, "desc", 1000, LocalDate.now(), LocalDate
96         .now().plusDays(100), null);
97     entity.setCategoryId(catId);
98
99     dao.saveOrUpdate(entity);
100    assertTrue(dao.get(entity.getId()) != null);
101    dao.delete(entity.getId());
102    assertTrue(dao.get(entity.getId()) == null);
103 }
104
105 }
```

OrderDaoTest.java

```
1 package at.lumetsnet.caas.test;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.assertTrue;
5
6 import java.time.LocalDate;
7 import java.time.LocalDateTime;
8 import java.util.Collection;
9
10 import org.junit.Before;
11 import org.junit.Test;
12
13 import at.lumetsnet.caas.dal.MenuCategoryDao;
14 import at.lumetsnet.caas.dal.MenuCategoryDaoJdbc;
15 import at.lumetsnet.caas.dal.MenuDao;
16 import at.lumetsnet.caas.dal.MenuDaoJdbc;
17 import at.lumetsnet.caas.dal.OrderDao;
18 import at.lumetsnet.caas.dal.OrderDaoJdbc;
19 import at.lumetsnet.caas.dal.UserDao;
20 import at.lumetsnet.caas.dal.UserDaoJdbc;
21 import at.lumetsnet.caas.model.Menu;
22 import at.lumetsnet.caas.model.MenuCategory;
23 import at.lumetsnet.caas.model.Order;
24 import at.lumetsnet.caas.model.User;
25
26 public class OrderDaoTest extends GenericDaoTest {
```



```
27
28 private OrderDao dao;
29 private MenuDao menuDao;
30 private MenuCategoryDao categoryDao;
31 private UserDao userDao;
32
33 private long menuId;
34 private long userId;
35
36 @Before
37 public void setUp() {
38     cleanTable("Order");
39     cleanTable("Menu");
40     cleanTable("MenuCategory");
41     cleanTable("User");
42
43     dao = new OrderDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
44     categoryDao = new MenuCategoryDaoJdbc(CONNECTION_STRING, USER_NAME,
45         PASSWORD);
46     menuDao = new MenuDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
47     userDao = new UserDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
48     MenuCategory cat = new MenuCategory(-1, "Cat1");
49     categoryDao.saveOrUpdate(cat);
50
51     Menu menu = new Menu(-1, "desc", 1000, LocalDate.now(), LocalDate.now()
52         .plusDays(100), null);
53     menu.setCategoryId(cat.getId());
54     menuDao.saveOrUpdate(menu);
55     menuId = menu.getId();
56
57     User user = new User(-1, "username", "pass", "firstname", "lastname",
58         false);
59     userDao.saveOrUpdate(user);
60     userId = user.getId();
61 }
62
63 @Test
64 public void insertTest() {
65
66     Order entity = new Order(-1, null, null, LocalDateTime.now(), "comment");
67     entity.setMenuId(menuId);
68     entity.setUserId(userId);
69     dao.saveOrUpdate(entity);
70     assertTrue(entity.getId() != -1);
71 }
72
73 @Test
74 public void updateTest() {
75     Order entity = new Order(-1, null, null, LocalDateTime.now(), "comment");
```

```
76     entity.setMenuId(menuId);
77     entity.setUserId(userId);
78     dao.saveOrUpdate(entity);
79     entity.setComment("new desc");
80     dao.saveOrUpdate(entity);
81     Order dbEntry = dao.get(entity.getId());
82     assertEquals("new desc", dbEntry.getComment());
83 }
84
85 @Test
86 public void getTest() {
87     Order entity = new Order(-1, null, null, LocalDateTime.now(), "comment");
88     // remove nano
89     entity.setTime(entity.getTime().minusNanos(entity.getTime().getNano()));
90
91     entity.setMenuId(menuId);
92     entity.setUserId(userId);
93     dao.saveOrUpdate(entity);
94
95     Order dbEntity = dao.get(entity.getId());
96     assertEquals(entity.getComment(), dbEntity.getComment());
97     assertEquals(entity.getMenuId(), dbEntity.getMenuId());
98     assertEquals(entity.getUserId(), dbEntity.getUserId());
99
100    assertEquals(entity.getTime(), dbEntity.getTime());
101
102 }
103
104 @Test
105 public void getAllTest() {
106     Order entity = new Order(-1, null, null, LocalDateTime.now(), "comment");
107     entity.setMenuId(menuId);
108     entity.setUserId(userId);
109     dao.saveOrUpdate(entity);
110     entity = new Order(-1, null, null, LocalDateTime.now(), "comment2");
111     entity.setMenuId(menuId);
112     entity.setUserId(userId);
113
114     dao.saveOrUpdate(entity);
115     assertEquals(2, dao.getAll().size());
116 }
117
118 @Test
119 public void deleteTest() {
120     Order entity = new Order(-1, null, null, LocalDateTime.now(), "comment");
121     entity.setMenuId(menuId);
122     entity.setUserId(userId);
123     dao.saveOrUpdate(entity);
124     assertTrue(dao.get(entity.getId()) != null);
```

```
125     dao.delete(entity.getId());
126
127     assertTrue(dao.get(entity.getId()) == null);
128 }
129
130 @Test
131 public void getOrdersByDateTest() {
132     Order entity = new Order(-1, null, null, LocalDateTime.now(), "today");
133     entity.setMenuId(menuId);
134     entity.setUserId(userId);
135     dao.saveOrUpdate(entity);
136     entity = new Order(-1, null, null, LocalDateTime.now().plusDays(1),
137         "other");
138     entity.setMenuId(menuId);
139     entity.setUserId(userId);
140     dao.saveOrUpdate(entity);
141
142     Collection<Order> result = dao.getOrdersByDate(LocalDate.now());
143     assertEquals(1, result.size());
144     assertEquals("today", result.stream().findFirst().orElse(null)
145         .getComment());
146 }
147
148 }
```

UserDaoTest.java

```
1 package at.lumetsnet.caas.test;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.assertTrue;
5
6 import org.junit.Before;
7 import org.junit.Test;
8
9 import at.lumetsnet.caas.dal.UserDao;
10 import at.lumetsnet.caas.dal.UserDaoJdbc;
11 import at.lumetsnet.caas.model.User;
12
13 public class UserDaoTest extends GenericDaoTest {
14
15     private UserDao dao;
16
17     @Before
18     public void setUp() {
19         cleanTable("Order");
20         cleanTable("Menu");
21         cleanTable("MenuCategory");
22         cleanTable("User");
```

```
23
24     dao = new UserDaoJdbc(CONNECTION_STRING, USER_NAME, PASSWORD);
25 }
26
27 @Test
28 public void insertTest() {
29
30     User entity = new User(-1, "username", "pass", "firstname", "lastname",
31         false);
32     dao.saveOrUpdate(entity);
33     assertTrue(entity.getId() != -1);
34 }
35
36 @Test
37 public void updateTest() {
38     User entity = new User(-1, "username", "pass", "firstname", "lastname",
39         false);
40     dao.saveOrUpdate(entity);
41     entity.setFirstName("newFirstName");
42     dao.saveOrUpdate(entity);
43     User dbEntry = dao.get(entity.getId());
44     assertEquals("newFirstName", dbEntry.getFirstName());
45 }
46
47 @Test
48 public void getTest() {
49     User entity = new User(-1, "username", "pass", "firstname", "lastname",
50         false);
51     dao.saveOrUpdate(entity);
52
53     User dbUser = dao.get(entity.getId());
54     assertEquals(entity.getUserName(), dbUser.getUserName());
55     assertEquals(entity.getPassword(), dbUser.getPassword());
56     assertEquals(entity.getFirstName(), dbUser.getFirstName());
57     assertEquals(entity.getLastName(), dbUser.getLastName());
58     assertEquals(entity.isLocked(), dbUser.isLocked());
59 }
60
61 @Test
62 public void getAllTest() {
63     User entity = new User(-1, "username", "pass", "firstname", "lastname",
64         false);
65     dao.saveOrUpdate(entity);
66     entity = new User(-1, "username1", "pass", "firstname1", "lastname1",
67         false);
68     dao.saveOrUpdate(entity);
69     assertEquals(2, dao.getAll().size());
70 }
71
```

```
72  @Test
73  public void deleteTest() {
74      User entity = new User(-1, "username", "pass", "firstname", "lastname",
75                          false);
76      dao.saveOrUpdate(entity);
77      assertTrue(dao.get(entity.getId()) != null);
78      dao.delete(entity.getId());
79      assertTrue(dao.get(entity.getId()) == null);
80  }
81
82 }
```

1.5 Sourcecode - CSS

main-window.css

```
1  /* styles used for all pages and the main window */
2  .nav-container {
3      -fx-border-width: 0 0 1 0;
4      -fx-border-color: black;
5      -fx-spacing: 10;
6      -fx-padding: 10;
7  }
8
9  .nav-username {
10     -fx-font: bold 10pt "Arial";
11 }
12
13 .nav-role {
14     -fx-font: 10pt "Arial";
15 }
16
17 .nav-command-container {
18     -fx-spacing: 10;
19     -fx-alignment: baseline-right;
20 }
21
22 .nav-command {
23     -fx-spacing: 10;
24     -fx-padding: 10;
25     -fx-font: 10pt "Arial";
26 }
27
28 .nav-command-selected {
29     -fx-background-color: #4CAF50;
30 }
31
32 #nav-top-icon {
```

```
33 -fx-padding: 10 0 0 0;
34 -fx-graphic: url('ic-info.png');
35 -fx-alignment: baseline-center;
36 -fx-content-display: center;
37 }
38
39 .table {
40 -fx-font: 10pt "Arial";
41 }
42
43 .table-command-container {
44 -fx-alignment: baseline-right;
45 -fx-spacing: 10;
46 }
47
48 .table-command {
49 -fx-background-position: center;
50 -fx-content-display: graphic-only;
51 -fx-min-height: 40px;
52 -fx-min-width: 40px;
53 }
54 .table-command-edit {
55 -fx-graphic: url('ic-edit.png');
56 }
57
58 .table-command-delete {
59 -fx-graphic: url('ic-delete.png');
60 }
61 .table-command-lock {
62 -fx-graphic: url('ic-lock.png');
63 }
64 .table-command-unlock {
65 -fx-graphic: url('ic-unlock.png');
66 }
67
68 .page-content-container {
69 -fx-padding: 10;
70 }
71
72 .page-title {
73 -fx-padding: 0 0 10 0;
74 -fx-font: 14pt Arial;
75 }
76
77 .page-command-container {
78 -fx-padding: 10;
79 -fx-alignment: baseline-right;
80 }
81 .page-command {
```

```
82  -fx-font: 10pt "Arial";
83  }
```

dialog.css

```
1  /* Styles used in Dialogs */
2  .dialog-top-container {
3      -fx-border-width: 0 0 1 0;
4      -fx-border-color: black;
5      -fx-spacing: 10;
6      -fx-padding: 10;
7  }
8
9  .dialog-bottom-container {
10     -fx-border-width: 1 0 0 0;
11     -fx-border-color: black;
12     -fx-spacing: 10;
13     -fx-padding: 15;
14 }
15
16 .dialog-title {
17     -fx-font: bold 12pt "Arial";
18 }
19
20 .dialog-description {
21     -fx-font: 10pt "Arial";
22 }
23
24 .command-button {
25     -fx-padding: 10;
26     -fx-font: 10pt "Arial";
27 }
28
29 .form-container {
30     -fx-padding: 10;
31 }
32
33 .form-label {
34     -fx-padding: 5 0 5 0;
35     -fx-font: 10pt "Arial";
36 }
37
38 .form-textfield {
39     -fx-font: 10pt "Arial";
40 }
41
42 .form-combobox {
43     -fx-font: 10pt "Arial";
44 }
```

```
45
46 .form-datepicker {
47   -fx-font: 10pt "Arial";
48 }
49
50 .error-label {
51   -fx-font: 10pt "Arial";
52   -fx-text-fill: Red;
53   -fx-padding: 10;
54   -fx-content-display: center;
55 }
```

derror-ialog.css

```
1 /* styles used for error dialog */
2 #dialog-icon {
3   -fx-padding: 10 0 0 0;
4   -fx-graphic: url('ic-error.png');
5   -fx-alignment: baseline-center;
6   -fx-content-display: center;
7 }
```

manage-menu-category-dialog.css

```
1 #dialog-icon {
2   -fx-padding: 10 0 0 0;
3   -fx-graphic: url('ic-edit.png');
4   -fx-alignment: baseline-center;
5   -fx-content-display: center;
6 }
```

manage-menu-dialog.css

```
1 /* dialog icon */
2 #dialog-icon {
3   -fx-padding: 10 0 0 0;
4   -fx-graphic: url('ic-edit.png');
5   -fx-alignment: baseline-center;
6   -fx-content-display: center;
7 }
```

manage-user-dialog.css

```
1 /* dialog icon */
2 #dialog-icon {
3   -fx-padding: 10 0 0 0;
```



```
4  -fx-graphic: url('ic-edit.png');  
5  -fx-alignment: baseline-center;  
6  -fx-content-display: center;  
7  }
```

1.6 Screenshots

Runs: 21/21 Errors: 0 Failures: 0

```
* at.lumetsnet.caas.test.MenuCategoryDaoTest [Runner: JUnit 4] (0.595 s)
  updateTest (0.491 s)
  getTest (0.039 s)
  getAllTest (0.026 s)
  insertTest (0.019 s)
  deleteTest (0.020 s)
* at.lumetsnet.caas.test.UserDaoTest [Runner: JUnit 4] (0.106 s)
  updateTest (0.024 s)
  getTest (0.025 s)
  getAllTest (0.020 s)
  insertTest (0.018 s)
  deleteTest (0.019 s)
* at.lumetsnet.caas.test.OrderDaoTest [Runner: JUnit 4] (0.295 s)
  getOrdersByDateTest (0.095 s)
  updateTest (0.035 s)
  getTest (0.036 s)
  getAllTest (0.034 s)
  insertTest (0.036 s)
  deleteTest (0.059 s)
* at.lumetsnet.caas.test.MenuDaoTest [Runner: JUnit 4] (0.129 s)
  updateTest (0.025 s)
  getTest (0.022 s)
  getAllTest (0.024 s)
  insertTest (0.029 s)
  deleteTest (0.029 s)
```

The screenshots show the CaaS web application interface. The left screenshot displays the 'Bereiche' (Areas) table with two rows of test data. The right screenshot displays the 'Hauptpreise' (Main Prices) table with two rows of test data. Both tables have columns for Name, Preis, Zeitraum, and Aktion. The interface includes a top navigation bar with 'Einstellungen', 'Benutzerverwaltung', and 'Hauptpreise' buttons. The user 'Roman Lumetsberger' is logged in as 'Administrator'.