Software Engineering & Projektmanagement

Ticketline 2.4 - Getting Started Guide

Version 2.4

(03. November 2015)

Dominik Moser, Manuel Mundorf, Wolfgang Gruber

Inhaltsverzeichnis

1 Einführung

Ticketline bildet das Grundgerüst für die Gruppenphase von SEPM. Es handelt sich dabei um ein vorkonfiguriertes Maven Projekt. Dies ermöglicht Ihnen die Entwicklung sofort mit einer vollständig lauffähigen Applikation zu beginnen, ohne sich mit dem Projektsetup beschäftigen zu müssen. Darüber hinaus wurden bereits einige Funktionen implementiert, um Ihnen das Zusammenspiel der einzelnen Komponenten zu demonstrieren. Der bereitgestellte Sourcecode kann selbstverständlich verändert, erweitert und angepasst werden.

Dieser Guide soll Ihnen bei den ersten Schritten mit Ticketline helfen. Er beginnt mit einer Installationsanleitung und stellt Ihnen im Anschluss die einzelnen Packages von Ticketline vor, sowie deren Anwendung innerhalb der Applikation.

Wir sind bemüht diesen Guide so gut wie möglich an die Bedürfnisse der Studierenden anzupassen. Dazu benötigen wir jedoch Ihre Hilfe. Wenn Sie Anregungen, Fehler oder Kritikpunkte gefunden haben, so zögern Sie bitte nicht uns diese mitzuteilen. Entweder über Ihren Tutor oder direkt per Mail an sepm@inso.tuwien.ac.at

Viel Spaß und gutes Gelingen mit Ticketline

2 Installation

2.1 Voraussetzungen

Voraussetzung für Ticketline ist die Installation des **JDK 8** (mindestens Update 20) [1] und **Maven 3.2.2** [2]. Weiterhin wird der JavaFX Scene Builder 2.0 build 20 [3] für die Entwicklung der GUI empfohlen.

Hinweis:
Wir empfehlen Java und Maven in Verzeichnisse zu installieren, die keine Leerzeichen enthalten, da hierdurch gelegentlich Probleme auftreten.

2.2 Java

Nachdem das JDK 8 heruntergeladen und installiert wurde, muss die *JAVA_HOME* Umgebungsvariable gesetzt werden (z.B. C:\Programme\Java\jdk1.8.0_11). Dies ist wichtig, da Maven die *JAVA_HOME* Umgebungsvariable verwendet. Zusätzlich muss die *PATH* Umgebungsvariable um den "JAVA_HOME"\bin Ordner erweitert werden (z.B. C:\Programme \Java\jdk1.8.0_11\bin).

2.3 Maven

Nachdem Sie die Maven Binaries heruntergeladen und in ein beliebiges Verzeichnis entpackt haben, müssen sie auch hier die *PATH* Umgebungsvariable erweitern. Genauso wie beim JDK muss der *bin* Ordner angegeben werden (z.B. C:\Programme\Maven\apache-maven-3.2.2\bin).

2.4 OS X

Wenn Sie unter Mac OS X entwickeln, prüfen Sie zuerst, ob Maven bereits installiert ist. Führen Sie hierzu den Befehl *mvn –version* im Terminal aus. Ab Mac OS X Version 10.9 (Maverick) wird Maven nicht mehr automatisch mitinstalliert. Um dies zu tun, laden Sie die Maven Binaries herunter und entpacken Sie sie in ein beliebiges Verzeichnis (z.B. /usr/local/apache-maven). Durch das Entpacken sollte das Unterverzeichnis *apache-maven-3.2.2* erstellt werden. Anschließend müssen die Umgebungsvariablen gesetzt werden. Erweitern Sie dazu die *.bash_profile*-Datei aus Ihrem User-Home-Verzeichnis um folgende Zeilen (siehe Listing 1).

[1] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

[2] http://maven.apache.org/download.cgi

[3] http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html

Listing 1: .bash_profile

```
1 export M2_HOME=/usr/local/apache-maven/apache-maven-3.2.2/
2 export M2=$M2_HOME/bin
3 export JAVA_HOME=$(/usr/libexec/java_home)
4 export PATH=$PATH:$M2:$JAVA_HOME/bin
```

Schließen Sie das Terminal und starten es erneut. Mit den Befehl *mvn –version* können Sie nun überprüfen, ob Maven korrekt installiert wurde.

2.5 Entwicklungsumgebung

Im Rahmen der LVA können Sie zwei verschiedene Entwicklungsumgebungen verwenden. Die kostenlose Open Source Eclipse IDE. Als Alternative bieten wir Ihnen in Zusammenarbeit mit JetBrains weiters die Möglichkeit die Ultimate Version der Entwicklungsumgebung IntelliJ IDEA zu nutzen.

2.5.1 Eclipse

Laden Sie die **Eclipse IDE for Java EE Developers** [4] herunter und installieren Sie sie in ein beliebiges Verzeichnis.

Anschließend können Sie in Eclipse über den Menüpunkt „Help“ und „Install New Software...“ noch das e(fx)clipse-Plugin installieren. Fügen Sie mittels „Add...“-Button die URL `http://download.eclipse.org/efxclipse/updates-released/1.2.0/site` hinzu und wählen Sie unter e(fx)clipse - install die Checkbox an. Die restlichen Checkboxen unter e(fx)clipse - single components können ignoriert werden, da sie automatisch installiert werden.

2.5.2 IntelliJ

Laden Sie **IntelliJ IDEA Ultimate Edition** [5] herunter und installieren Sie sie in ein beliebiges Verzeichnis.

Im TUWEL-Kurs finden Sie einen Lizenzschlüssel, den Sie im Rahmen der LVA verwenden dürfen.

[4] `http://www.eclipse.org/downloads/`
[5] `http://www.jetbrains.com/idea/download/`

2.6 Ticketline

Als ersten Schritt müssen Sie eine lokale Version des Ticketline Projekts anlegen. Laden Sie dazu das Ticketline Projekt aus TUWEL herunter und entpacken Sie es in ein Verzeichnis Ihrer Wahl.

Starten Sie nun die gewählte Entwicklungsumgebung und importieren Sie das Projekt wie folgt:

- **Import in Eclipse:**
  Wählen Sie dazu *File* → *Import...* → *Maven* → *Existing Maven Projects*. Wählen Sie das Ticketline-Hauptverzeichnis aus und importieren Sie das Projekt mit *Finish*.

- **Import in IntelliJ:**
  Wählen Sie im *Startbildschirm* → *Import Project*. Wählen Sie das Ticketline-Hauptverzeichnis aus und klicken Sie auf *OK* Im nächsten Bildschirm wählen Sie *Import project from external model* → *Maven* → *Next*. Im anschließenden Fenster wählen Sie *Search for projects recursively*, sowie *Import Maven projects automatically* und *Create module groups for multi-module Maven projects*. Die restlichen Einstellungen können Sie beibehalten. Alle weiteren Punkte können Sie mit *Next* bestätigen.

Hinweis:
Nach Änderungen im DTO- und Database-Projekt oder falls Maven Fehler beim Auflösen der Abhängigkeiten zwischen den Ticketline-Projekten haben sollte und der Build-Vorgang nicht erfolgreich abschließt, wechseln Sie mit der Kommandozeile in das Ticketline Hauptverzeichnis und führen Sie den folgenden Befehl aus: (siehe Listing 2)

Listing 2: Maven project clean install

```
1 mvn clean install
```

2.6.1 Datenbank

Starten Sie nun die Ticketline-Datenbank. Öffnen Sie dazu die Kommandozeile und wechseln in das *database*-Verzeichnis. Führen Sie dann folgenden Befehl aus (siehe Listing 3).

Listing 3: Maven run h2-Database

```
1 mvn exec:java -P h2
```

Die Datenbank wird dabei im Unterverzeichnis *db* erstellt.
Gleichzeitig öffnet sich ein Browserfenster und es wird automatisch eine Verbindung zum integrierten Datenbankmanager aufgebaut.

2.6.2 Data Generator

Im nächsten Schritt müssen Sie das Ticketline Data Generator-Projekt ausführen. Öffnen Sie dazu die Kommandozeile und wechseln in das *dataGenerator*-Verzeichnis. Führen Sie dann folgenden Befehl aus (siehe Listing 4).

Listing 4: Maven run dataGenerator

```
1 mvn exec:java -P generateData
```

Beim Ausführen wird die Datenbank sowohl initialisiert als auch die erste Daten in die Datenbank geschrieben. Es ist zu beachten, dass bei jeder Ausführung des Data Generator das Datenbankschema gelöscht und neu erstellt wird. Es gehen somit alle Änderungen an den Daten, sowie neu hinzugefügte Datensätze verloren. Ist dies nicht gewünscht, muss im *dataGenerator*-Projekt in src\main \resources \spring \datagenerator-context.xml (siehe Listing 5), der Wert von **create-drop** auf **update** oder **validate** geändert werden.

Listing 5: datagenerator-context.xml

```
25 <prop key="hibernate.hbm2ddl.auto">create-drop</prop>
```

2.6.3 Server

Um nun den Ticketline-Server zu starten, wechseln Sie in der Kommandozeile in das *server*-Verzeichnis (*server*). Dort führen Sie folgendes Kommando aus (siehe Listing 6).

Listing 6: Maven run ticketline server

```
1 mvn jetty:run
```

Dadurch werden alle benötigten Bibliotheken heruntergeladen und ein Jetty Server gestartet.

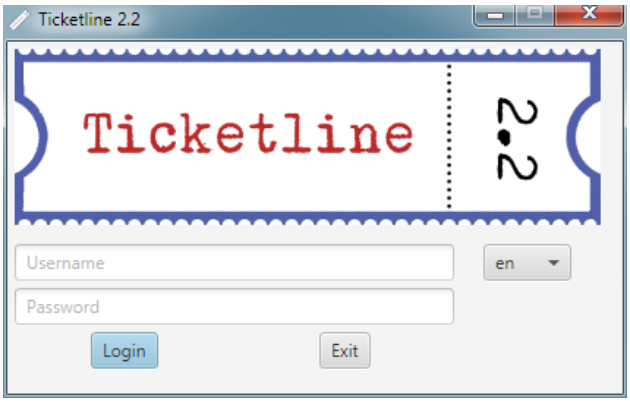

Abbildung 1: Ticketline Client Login

2.6.4 Client

Zuletzt kann nun das Client-Projekt gestartet werden. Rechtsklicken Sie dazu in Eclipse/IntelliJ auf die *TicketlineClient-Klasse* im Package *at.ac.tuwien.inso.ticketline.client* des *client* Projekts und dann auf *Run As* → *Java Application* beziehungsweise in IntelliJ auf *Run TicketlineClient... main()*.
Alternativ können Sie den Client auch über die Kommondozeile aus dem *client*-Verzeichnis starten (siehe Listing 7).

Listing 7: Maven run ticketline client

```
1 mvn exec:java -P client
```

Das in Abbildung 1 dargestellte Fenster sollte nun erscheinen. Mit "**marvin**" als Benutzernamen und "**42**" als Passwort und dem Klick auf *Login* öffnet sich das Hauptfenster der Anwendung.
Hinweis:
Falls hier die Fehlermeldung erscheint, dass der Server nicht erreicht werden konnte, überprüfen Sie ob sowohl die Datenbank als auch der Server fehlerfrei gestartet wurden. Listing 2)

3 Frameworks & Bibliotheken

Wie jedes größere Projekt in Java setzt auch Ticketline eine Reihe von Frameworks und Bibliotheken ein. Die nachfolgende Tabelle stellt sie kurz vor:

Name	**Kurzbeschreibung**
Spring	Framework für die Entwicklung von Java/Java EE Anwendungen Website: `http://spring.io`
Hibernate	Object/Relational Mapping Bibliothek Website: `http://hibernate.org/`
H2	Java SQL Datenbank Engine Website: `http://www.h2database.com/`
c3p0	JDBC Connection Pool Website: `http://www.mchange.com/projects/c3p0/`
SLF4J	Logging Bibliothek Website: `http://www.slf4j.org/`
Jackson	Dient zur Umwandlung von Java Objekten in JSON und umgekehrt Website: `http://wiki.fasterxml.com/JacksonHome`
HttpClient	HTTP-Client von Apache Website: `https://hc.apache.org/httpcomponents-client-4.3.x/index.html`
JavaFX	Framework für die Entwicklung von plattformübergreifende Rich Client Applications. Wird bei Ticketline für die Entwicklung des User Interfaces verwendet. Website: `http://docs.oracle.com/javafx/`
ControlsFX	UI Komponenten Bibliothek Website: `http://fxexperience.com/controlsfx/`
JUnit	Testing Framework Website: `http://junit.org`
Mockito	Mocking Framework Website: `http://code.google.com/p/mockito/`
Swagger	Dokumentation der REST-Services Website: `https://github.com/swagger-api/swagger-core/wiki/Annotations`

Tabelle 1: Frameworks & Bibliotheken

4 Struktur von Ticketline

Ticketline gliedert sich in die folgenden fünf Projekte/Module:

- Client Projekt: client
- Datenbank Projekt: database
- Daten Generator Projekt: dataGenerator
- DTO Projekt: dto
- Server Projekt: server

Diese Projekte werden in diesem Kapitel genauer beschrieben. Die Abhängigkeiten zwischen den einzelnen Projekten sind in Abbildung 2 dargestellt.

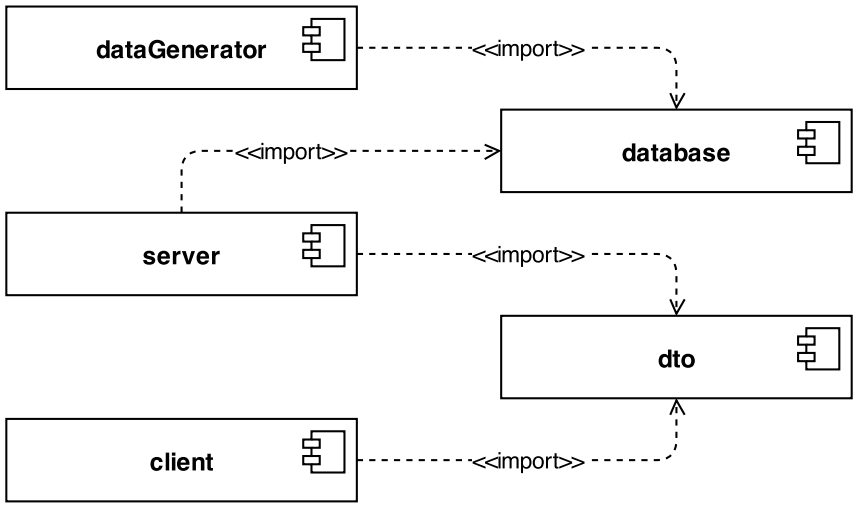

Abbildung 2: Ticketline Abhängigkeiten

4.1 Ticketline Server

In diesem Kapitel werden das Server-Projekt und dessen Packages genauer beschrieben, sowie deren Struktur und Verwendung erläutert. Abbildung 3 zeigt alle Packages des Servers und deren Abhängigkeiten.
Die Abhängigkeiten zum Datenbank-Projekt sind hier aus Übersichtsgründen nicht dargestellt.
Spring bietet mehrere Varianten zur Konfiguration einer Applikation an, wobei in Ticketline die Konfiguration über XML-Konfigurationsdateien und Annotationen erfolgt.

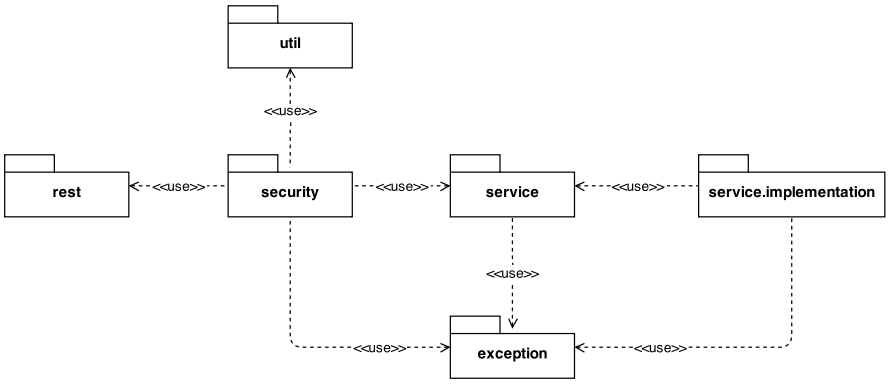

Abbildung 3: Ticketline Server Packages

Folgende Konfigurationsdateien sind für den Ticketline Server relevant:

server-context.xml
In der *server-context.xml* wird die *db-context.xml* geladen, die in 4.5 genauer beschrieben wird. Des Weiteren wird das Package angegeben, in dem Spring beim Start-Up nach Komponenten sucht, die dann automatisch initialisiert werden.
Die *server-context.xml* ist unter src\main\resources\spring zu finden.

spring-security.xml
In der *spring-security.xml* werden Sicherheitseinstellungen vorgenommen. Besonders wichtig sind hierbei die *intercept-url* Einträge. Folgende zwei Einträge sind dabei zu beachten (siehe Listing 8).

Listing 8: spring-security.xml

```
30 <intercept-url pattern="/login" access="permitAll"/>
31 <intercept-url pattern="/service/info/user" access="permitAll"/>
32 <intercept-url pattern="/**" access="isAuthenticated()"/>
```

Die erste und zweite Zeile erlauben den anonymen Zugriff auf das Login und die Benutzerinformationen, die angemeldeten Benutzers. In der dritten Zeile wird der Zugriff auf alle weiteren REST-Services des Servers auf angemeldete Benutzer beschränkt.
Die *spring-security.xml* befindet sich im Verzeichnis src\main\webapp\WEB-INF\spring

test-context.xml
Die *test-context.xml* enthält Einstellungen für die Testklassen. Dabei besteht der wesentliche Unterschied darin, dass nicht die Produktivdatenbank verwendet wird, sondern eine Testda-

tenbank. Bei dieser Testdatenbank handelt es sich um eine In-Memory-Datenbank. Sie wird bei jedem Testlauf neu initialisiert und mit den Testdaten befüllt (siehe Listing 9).

Listing 9: Hibernate load testdata

```
38 <prop key="hibernate.hbm2ddl.import_files">sql/test_dataScript.sql</
      prop>
```

Mit dieser Zeile wird das Script für die Testdaten definiert. Es befindet sich in src\main\resources\sql und muss für weitere Testfälle entsprechend erweitert werden.

4.1.1 REST Package

Das Package *at.ac.tuwien.inso.ticketline.server.rest* enthält die server-seitigen REST-Schnittstellen, die vom Ticketline-Client aufgerufen werden. Jeder Request an den Server führt zu einem Aufruf einer der Methoden, die mit der @RequestMapping-Annotation versehen sind (siehe Listing 10).

Listing 10: Beispiel News Request Mapping

```
1 @RestController
2 @RequestMapping(value = "/news")
3 public class NewsController {
4
5   @RequestMapping(value = "/{id}", method = RequestMethod.GET,
        produces = MimeTypeUtils.APPLICATION_JSON_VALUE)
6   public NewsDto getNewsById(@PathVariable("id") Integer id) throws
        ServiceException {
7     ...
```

Mit der @RequestMapping-Annotation wird also das Mapping zwischen URL und Funktion definiert. Durch den *method*-Parameter können die gültigen HTTP-Request-Arten festgelegt werden (GET, POST, PUT, DELETE). Weiterhin können mit @RequestParam Funktionsparameter definiert werden. Auch andere Arten der Parameterübergabe sind möglich. Weitere Informationen dazu finden Sie in der Spring Dokumentation [6].
Zwei weitere Annotationen sind im Controller zu beachten (siehe Listing 11).

[6] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

Listing 11: Weitere Spring Annotationen im NewsController

```
1 @RestController
2 @RequestMapping(value = "/news")
3 public class NewsController {
4
5   @Autowired
6   private NewsService newsService;
```

Jede Controller-Klasse muss mit *@RestController* annotiert werden, da Spring diese Klasse nur so beim Starten des Jetty Servers instanziiert.
@Autowired bewirkt das Injecten der *newsService*-Property. Dies funktioniert jedoch nur, wenn die Service Klasse mit *@Service* annotiert ist.

4.1.2 Service Package

Im Package *at.ac.tuwien.inso.ticketline.server.service* werden die Interfaces für die Service-Klassen definiert und das Sub-Package *implementation* enthält dementsprechend die Implementierungen dieser Interfaces (siehe Listing 12).

Listing 12: Spring Annotationen im NewsService

```
1 @Service
2 public class SimpleNewsService implements NewsService {
3
4   @Autowired
5   private NewsDao newsDao;
```

Jede Service-Implementierung muss mit *@Service* annotiert sein. Andernfalls wird die Klasse nicht beim Starten des Jetty-Servers instanziert und das Injecten im Controller würde nicht funktionieren. Auch im Service ist wieder die *@Autowired*-Annotation zu finden. Hierdurch wird die DAO, die für den Datenbankzugriff benötigt wird, injectet. Dies funktioniert nur, wenn die DAO Implementierung mit *@Repository* annotiert ist. Die DAOs sind jedoch nicht im Server Projekt, sondern im Datenbank Projekt enthalten.

4.2 Ticketline Client

In diesem Kapitel wird das Client-Projekt und dessen Packages genauer beschrieben, sowie deren Struktur und Verwendung erläutert. Abbildung 4 zeigt alle Packages des Clients und deren Abhängigkeiten.
Die Abhängigkeiten zum DTO-Projekt sind hier aus Übersichtsgründen nicht dargestellt.

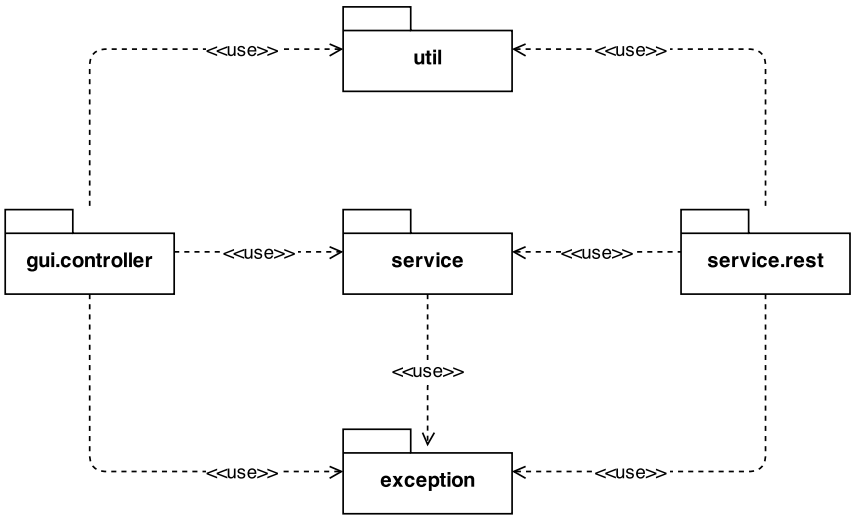

Abbildung 4: Ticketline Client Packages

4.2.1 Controller Package

Im Package *at.ac.tuwien.inso.ticketline.client.gui.controller* sind die GUI-Controller enthalten. Im Projekt sind bereits der *LoginController*, der *NewsBoxController* und der *NewsElementController* vorhanden. Die Controller müssen in den FXML-Dateien angegeben werden, die sich in src\main\resources\gui\fxml befinden. Weiterhin müssen Controller das *Initializable*-Interface implementieren. Mit der *@FXML*-Annotation kann auf Felder, die in der FXML-Datei definiert wurden, zugegriffen werden (siehe 13).

Listing 13: @FXML Annotation

```
1 @Component
2 public class LoginController implements Initializable {
3
4   @FXML
5   private TextField txtUsername;
```

4.2.2 Client Package

Im Package *at.ac.tuwien.inso.ticketline.client.service* werden die Interfaces für die Kommunikation mit dem Server definiert. Die Implementierungen sind im Sub-Package *implementation* zu finden. Es ist bereits das Login, das Logout, sowie das Abrufen der News implementiert. Die URLs für die Verbindung zum Server werden aus der *ticketlineClientConfig.properies*-Datei ausgelesen. Diese befindet sich in src\main\resources\config und muss ggf. angepasst und erweitert werden.

4.2.3 Util Package

Das Package *at.ac.tuwien.inso.ticketline.client.util* enthält die Klassen *BundleManager* und *SpringFxmlLoader*. Die *BundleManager*-Klasse wird verwendet um die Bundles für die Internationalisierung einheitlich im Projekt zur Verfügung zu stellen.
Die Property-Dateien, die zur Internationalisierung verwendet werden, finden Sie in src\main\resources\localization. Diese müssen während der Entwicklung erweitert werden.
Der *SpringFxmlLoader* wird im Ticketline-Client zum Laden der FXML-Dateien verwendet und sorgt dafür, dass mittels *@Inject*-annotierte Properties gesetzt werden. Wenn eine FXML-Datei geladen werden soll, muss dies daher immer über den *SpringFxmlLoader* geschehen, da nur dadurch die Spring-Beans mittels Dependency Injection im jeweiligen Controller gesetzt werden. Listing 14 zeigt die Verwendung des *SpringFxmlLoader*.

Listing 14: Verwendung des SpringFxmlLoader im LoginController

```
1 @Component
2 public class LoginController implements Initializable {
3
4   @Autowired
5   private SpringFxmlLoader springFxmlLoader;
6
7   @FXML
8   private void handleLogin(ActionEvent event) {
9   ...
10  newsStage.setScene(new Scene((Parent) springFxmlLoader.load("/gui/
        fxml/newsBox.fxml")));
```

4.3 Ticketline Data Generator

Das Data Generator-Projekt soll für die Generierung von Daten verwendet werden. Es besteht lediglich aus einem Package, dass in Abbildung 5 dargestellt ist. Es enthält bereits Generatoren für News und Employees.

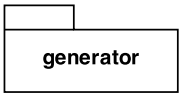

Abbildung 5: Ticketline Data Generator Packages

Genauso wie im Server wird auch in diesem Projekt Spring und Hibernate verwendet. Da die Anwendung jedoch nicht in einem Jetty-Server läuft, muss der *Application Context* [7] manuell geladen werden. Dies geschieht in der *DataGeneratorMain*-Klasse (siehe Listing 15 Zeile 1)

Listing 15: Laden des Application Context und Beans

```
1 ClassPathXmlApplicationContext context =
2   new ClassPathXmlApplicationContext("/spring/datagenerator-context.
        xml");
3 ...
4 context.getBean(EmployeeGenerator.class).generate();
5 context.getBean(NewsGenerator.class).generate();
```

Weiterhin können die Generatoren nicht durch *@Autowired* injected werden, sondern müssen manuell aus dem ApplicationContext ausgelesen werden (siehe Listing 15 Zeile 3 und Zeile 4).

4.3.1 Generator Package

Im Package *at.ac.tuwien.inso.ticketline.datagenerator.generator* sind die eigentlichen Generatoren enthalten. Wie bereits erwähnt sind bisher nur die Generatoren für News und Angestellte implementiert.
Es ist zu beachten, dass die Generatoren das *DataGenerator*-Interface implementieren, die DAOs mit der *@Autowired* Annotation injectet werden können und die Generatoren selbst mit *@Component* annotiert werden müssen.
Die DAOs sind, genauso wie für den Server, im Datenbank-Projekt enthalten.

[7] http://docs.spring.io/spring/docs/4.1.5.RELEASE/javadoc-api/org/springframework/context/ApplicationContext.html

4.4 Ticketline DTO

Das DTO-Projekt enthält alle Klassen, die für den Austausch von Daten zwischen Client und Server benötigt werden. Im Laufe der Entwicklung ist dieses Projekt durch die zusätzlich benötigten DTOs zu erweitern.

4.5 Ticketline Datenbank

Das Datenbank-Projekt enthält alle Klassen die zum OR-Mapping zwischen Datenbank und Server bzw. Data Generator benötigt werden. Es enthält sowohl die Model-Klassen als auch die dazugehörigen DAOs.

4.5.1 Model Package

Dieses Package enthält die Entities für Ticketline. Im Gegensatz zu den anderen Projekten enthält das Model bereits alle benötigten Klassen und nicht nur die Klassen, die für den Beispiel Anwendungsfall relevant sind. Falls jedoch während der Entwicklung zusätzliche Entities bzw. Tabellen benötigt werden, ist dieses Package zu erweitern.

4.5.2 DAO Package

Das Package *at.ac.tuwien.inso.ticketline.db.dao* enthält bisher die Klassen EmployeeDao und NewsDao, beide Klassen erweitern JpaRepository[8]. *JpaRepository* ist eine vom Spring Framework bereitgestellte Klasse, die CRUD-Funktionalitäten[9] zur Verfügung stellt. Weiters müssen die DAOs mit *@Repository* annotiert werden. Um die DAOs mit einfachen Abfragen zu erweitern, kann die JPA Criteria API verwendet werden. Ein Beispiel dazu findet sich in der *EmployeeDao* (siehe Listing 16).

Listing 16: JPA Criteria API Beispiel EmployeeDao

```
1 @Repository
2 public interface EmployeeDao extends JpaRepository<Employee, Integer>
      {
3   public List<Employee> findByUsername(String username);
4 }
```

findByUsername liefert alle Angestellten mit dem übergebenen Benutzernamen. Wichtig dabei ist, dass die *Employee*-Entity eine Property names “username“ enthält. Kompliziertere

[8] http://docs.spring.io/spring-data/jpa/docs/1.7.x/api/org/springframework/data/jpa/repository/JpaRepository.html

[9] http://de.wikipedia.org/wiki/CRUD

Abfragen lassen sich mittels JPQL und der *@Query* Annotation definieren. Weitere Informationen zu beiden Varianten finden Sie in der JPA Repository-Dokumantation[10].

Weiterhin sind die Konfigurationsdateien zu beachten.
Diese befinden sich in src\main\resources\spring

db-context.xml
Die *db-context.xml* Datei lädt im wesentlichen nur die Konfigurationsdateien *db-context-dao.xml* und *db-context-database.xml*.

db-context-dao.xml
Die *db-context-dao.xml* definiert das Package, in dem die DAOs enthalten sind und sorgt somit dafür, dass Spring die DAOs in den unterschiedlichen Klassen injecten kann.

db-context-database.xml
Die *db-context-database.xml* enthält die Verbindungseinstellungen zur Datenbank und definiert den TransactionManager, sowie die EntityManagerFactory.

[10] `http://docs.spring.io/spring-data/jpa/docs/1.7.1.RELEASE/reference/html/`

5 FAQ

5.1 Was darf an Ticketline geändert werden?

Sie können Ticketline nach Belieben ändern und an Ihre Bedürfnisse anpassen.

Die bestehende Struktur und Code sollen Sie dabei unterstützen, weniger Zeit mit dem Projekt-Setup und der Definition der Strukturen zu verbringen und somit schneller mit der Umsetzung der eigentlichen Anforderungen beginnen zu können.

Weiters handelt es sich bei der bestehenden Applikation um einen sogenannten *Durchstich*, d.h. die Applikation enthält einen kompletten Use Case, der einen Zugriff vom User Interface durch alle Schichten bis zur Datenbank zeigt.

5.2 Dürfen zusätzliche Bibliotheken verwendet werden?

Sollten Sie zusätzlich Bibliotheken benötigen, binden Sie diese über Maven ein. Achten Sie hierbei auf die jeweiligen Lizenzbestimmungen.

5.3 Ich kenne mich mit Technologie-XY nicht aus, was nun?

An erster Stelle steht hier die eigene Recherche. Lesen Sie die Dokumentation der betreffenden Technologie und suchen Sie im Internet (z.B. auf http://stackoverflow.com) nach Lösungen für Ihr Problem. Sollten Sie danach nicht weiterkommen, wird Ihr Tutor Ihnen beratend zur Seite stehen.