

Software Engineering & Projektmanagement

Ticketline 3.1 - Getting Started Guide

Version 3.1
(01. April 2018)

Thomas Artner, Wolfgang Gruber

Inhaltsverzeichnis

1	Einführung	3
2	Installation	4
2.1	Voraussetzungen	4
2.2	Java	4
2.3	Entwicklungsumgebung	4
2.4	Ticketline	4
2.4.1	Server	5
2.4.2	Datenbank	5
2.4.3	Data Generator	5
2.4.4	Client	6
3	Frameworks & Bibliotheken	8
4	Struktur von Ticketline	9
4.1	Ticketline Server	9
4.1.1	Endpoint Package	10
4.1.2	Swagger Annotations & SwaggerUI	12
4.1.3	Service Package	15
4.1.4	Entity Package	16
4.1.5	Repository Package	16
4.2	Ticketline Client	16
4.2.1	Controller Package	17
4.2.2	Service Package	18
4.2.3	REST Package	18
4.2.4	Util Package	18
4.3	Ticketline REST	19
5	FAQ	20
5.1	Was darf an Ticketline geändert werden?	20
5.2	Dürfen zusätzliche Bibliotheken verwendet werden?	20
5.3	Ich kenne mich mit Technologie-XY nicht aus, was nun?	20

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 10** ¹. Weiterhin wird der JavaFX Scene Builder 9.0.1+ ² für die Entwicklung der GUI empfohlen sowie Scenic View ³ empfohlen. Weiters wird als Build und Dependency Management Tool Apache Maven ⁴ verwendet. Ticketline liefert hier bereits eine "gebundene" Version mit.

2.2 Java

Nachdem das JDK 10 heruntergeladen und installiert wurde, muss die `JAVA_HOME` Umgebungsvariable gesetzt werden (z.B. `C:\Programme\Java\jdk-10_xx`). 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\jdk-10_xx\bin`).

2.3 Entwicklungsumgebung

Im Rahmen der LVA sollen Sie eine moderne Entwicklungsumgebung kennenlernen. Wir verwenden hier IntelliJ von IDEA. Diese steht in einer freien Community Edition und einer kostenpflichtigen Ultimate Edition zur Verfügung.

Laden Sie **IntelliJ IDEA** ⁵ herunter und installieren Sie sie in ein beliebiges Verzeichnis.

Für Studierende gibt es die Möglichkeit eine kostenfreie Educational License zu beantragen ⁶ die sie während ihres Studiums verwenden können. Sollten Sie die Community Edition bevorzugen ist dies auch möglich.

2.4 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. Nach dem Start der Entwicklungsumgebung importieren Sie das Projekt wie folgt: Wählen Sie im *Startbildschirm* → *Open Project* und öffnen Sie das Ticketline-Hauptverzeichnis.

¹<https://www.oracle.com/technetwork/java/javase/downloads/>

²<https://gluonhq.com/labs/scene-builder/>

³<http://fxexperience.com/scenic-view/>

⁴<https://maven.apache.org/>

⁵<https://www.jetbrains.com/idea/download/>

⁶<https://www.jetbrains.com/student/>

2.4.1 Server

Um nun den Ticketline-Server zu starten, wechseln Sie in der Kommandozeile in das *ticketline*-Verzeichnis. Dort führen Sie folgendes Kommando aus (siehe Listing 1).

Listing 1: Maven run ticketline server

```
1 ./mvnw -pl=server -am spring-boot:run
```

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

Der Prozess darf nicht abgebrochen und das Kommandozeilen-Fenster darf nicht geschlossen werden, da sonst der Server beendet wird.

Hinweis: unter Windows kann sich je nach verwendetet Windows Version und Terminal das Kommando leicht unterscheiden, unter Umständen müssen sie statt “./mvnw” nur “mvnw” schreiben oder auch “mvnw.cmd”

2.4.2 Datenbank

Beim starten des Ticketline Servers startet automatisch eine Datenbank mit. Die Datenbank wird im Unterverzeichnis *./database/* erstellt und sollte nie mit eingecheckt werden.

Der Server ist so konfiguriert, dass versucht wird das Datenbmodell in der Datenbank entsprechend Ihrem Datenmodell upzudaten. Da das nicht immer funktioniert kann es sein, dass sie manuell die Datenbank löschen müssen. Sie wird dann automatisch beim nächsten Start neu erstellt.

Mit der Datenbank startet auch ein Webserver der ein einfaches Verbinden zur Datenbank ermöglicht. Über die URL `http://localhost:8080/h2-console`, können Sie eine Verbindung zum integrierten Datenbankmanager aufbauen.

Geben Sie im jdbc connection String den absoluten Pfad zu Ihrer Datenbank an:

jdbc:h2:file:<absolutePathToTicketline>/database/ticketline

Als Username sowie Passwort verwenden Sie *ticketline*.

2.4.3 Data Generator

Ticketline bietet die Möglichkeit Demo Daten zu generieren sollten diese beim starten noch nicht bestehen. Dies wird über eigene DataGenerator Klassen erreicht. Starten Sie dazu Ihren Server mit den entsprechendem Profil (siehe Listing 2).

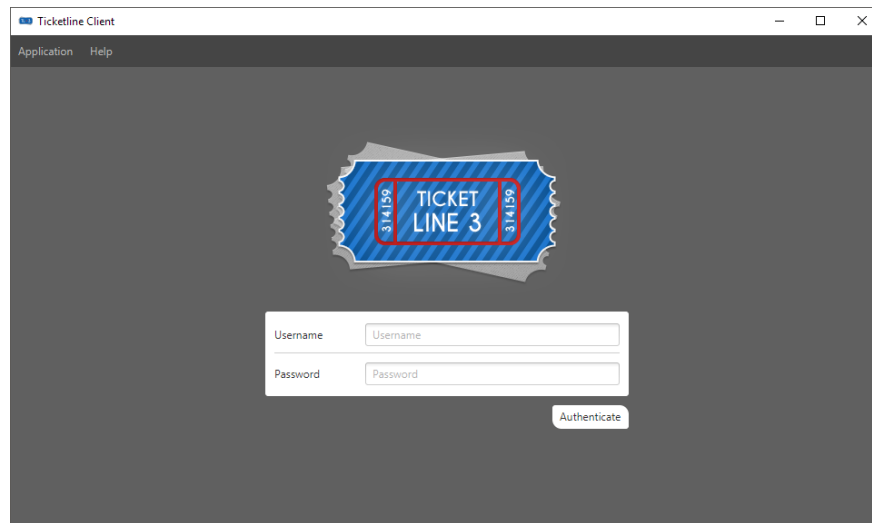


Abbildung 1: Ticketline Client Login

Listing 2: Maven run ticketline server with demo data

```
1 ./mvnw -pl=server -am spring-boot:run -Dspring-boot.run.profiles=generateData
```

2.4.4 Client

Zuletzt kann nun das Client-Projekt gestartet werden. Rechtsklicken Sie dazu in IntelliJ auf die *TicketlineClientApplication-Klasse* im Package *at.ac.tuwien.inso.sepm.ticketline.client* des *client* Projekts und dann auf *Run TicketlineClientApplication...*

Alternativ können Sie den Client auch über die Kommandozeile aus dem *ticketline*-Verzeichnis starten (siehe Listing 3).

Listing 3: Maven run ticketline client

```
1 ./mvnw -pl=client -am spring-boot:run
```

Das in Abbildung 1 dargestellte Fenster sollte nun erscheinen.

Mit “**admin**“ als Benutzernamen und “**password**“ als Passwort und dem Klick auf *Login* können Sie sich als Administrator anmelden.

Für eine Anmeldung als user verwenden Sie “**user**“ als Benutzernamen und “**password**“ als Passwort.

Hinweis: Falls hier die Fehlermeldung erscheint, dass der Server nicht erreicht werden kann-

te, überprüfen Sie ob der Server fehlerfrei gestartet wurden.

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/
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/
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
Swagger Springfox	Dokumentation der REST-Services Website: http://springfox.github.io/springfox/

Tabelle 1: Frameworks & Bibliotheken

4 Struktur von Ticketline

Ticketline gliedert sich in die folgenden fünf Projekte/Module:

- Rest Projekt: rest
- Client Projekt: client
- Server Projekt: server

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

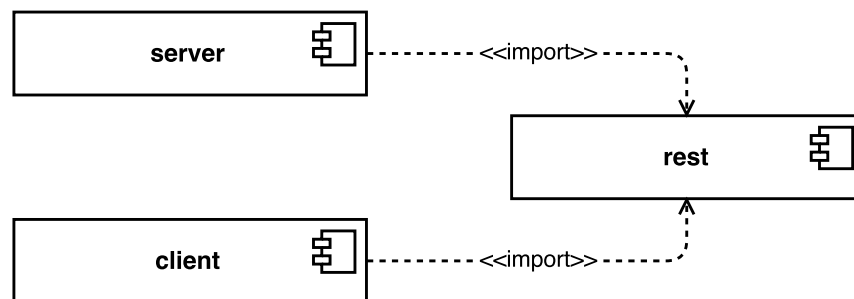


Abbildung 2: Ticketline Abhängigkeiten

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.

Spring bietet mehrere Varianten zur Konfiguration einer Applikation an, wobei in Ticketline die Konfiguration über Annotationen erfolgt.

Folgende Konfigurationsdateien (im *YAML*-Format) sind für den Ticketline Server relevant:

application.yml

In der *application.yml* werden alle Standardkonfigurationen eingetragen. Das beinhaltet zum Beispiel Einstellungen zum Logging aber auch Informationen zur Datenbankverbindung oder zur Anwendungssicherheit.

Spring kann weitere Konfigurationsdateien je nach angegebenen Profil laden. Für ein *test*-Profil würde die Konfigurationsdatei dann beispielsweise *application-test.yml* heißen.

@Configuration-Klassen

Zusätzlich zur Konfiguration mit *YAML*-Files werden so genannte Configuration-Klassen verwendet. Das sind java-Klassen die mit der *@Configuration*-Annotation versehen sind. Spring

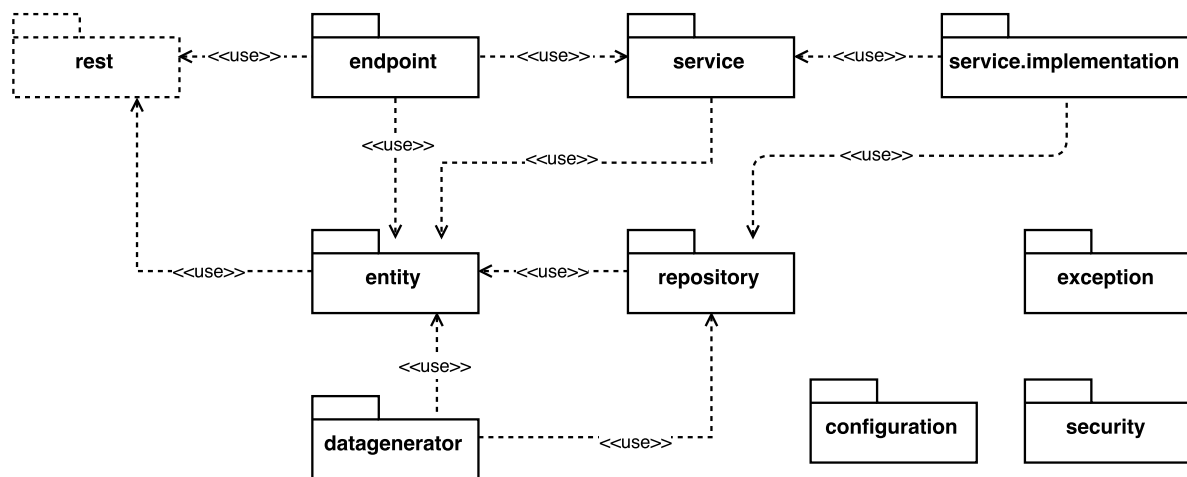


Abbildung 3: Ticketline Server Packages

erkennt diese Klassen automatisch und führt Sie zum Start der Anwendung aus. Hier wird beispielsweise die JSON Darstellung der Objekte aber auch die Applikationssicherheit und die Swagger Dokumentation konfiguriert. In der Security Configuration können beispielsweise neue User eingefügt werden.

4.1.1 Endpoint Package

Das Package *at.ac.tuwien.inso.sepm.ticketline.server.endpoint* 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 4).

Listing 4: Beispiel News Request Mapping

```
1 @RestController
2 @RequestMapping(value = "/news")
3 @Api(value = "news", description = "News_REST_service")
4 public class NewsEndpoint {
5
6     // ...
7
8     @RequestMapping(method = RequestMethod.GET)
9     @ApiOperation(value = "Get_list_of_simple_news_entries")
10    public List<SimpleNewsDTO> findAll() {
11        return newsMapper.newsToSimpleNewsDTO(newsService.findAll());
12    }
13
14    // ...
15
16 }
```

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* und *@PathVariable* Funktionsparameter definiert werden. Auch andere Arten der Parameterübergabe sind möglich. Weitere Informationen dazu finden Sie in der Spring Dokumentation⁷.

Jede Controller-Klasse muss mit *@RestController* annotiert werden, da Spring diese Klasse nur so beim Starten des Servers instanziiert.

Abhängigkeiten die von der Klasse benötigt werden, werden im Konstruktor mitgegeben. Diese werden dann von Spring automatisch bei der Instanziierung der Klasse mitgegeben. Dazu dürfen nur Klassen mitgegeben werden die Spring bereits bekannt sind. Im Codebeispiel (siehe Listing 5) wird beispielsweise eine Instanz des *NewsService* und eine Instanz des *NewsMapper* im Konstruktor angefordert. Da diese Beiden Klassen Spring bereits bekannt sind werden Sie beim Instanzieren von *NewsEndpoint* automatisch übergeben.

⁷<http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-developing-web-applications.html>

Listing 5: Konstruktor Parameter

```
1 @RestController
2 @RequestMapping(value = "/news")
3 @Api(value = "news", description = "News_REST_service")
4 public class NewsEndpoint {
5
6     private final NewsService newsService;
7     private final NewsMapper newsMapper;
8
9     public NewsEndpoint(NewsService newsService, NewsMapper newsMapper)
10     {
11         this.newsService = newsService;
12         this.newsMapper = newsMapper;
13     }
14     // ...
15
16 }
```

4.1.2 Swagger Annotations & SwaggerUI

Im Package *at.ac.tuwien.inso.sepm.ticketline.server.endpoint* finden sich zudem auch Annotations des Swagger Frameworks⁸, mit deren Hilfe wird automatisiert eine interaktive Dokumentation der REST-Schnittstelle erstellt.

Achten Sie in Listing 6 besonders auf die Annotations *@Api(...)* und *@ApiOperation(...)*.

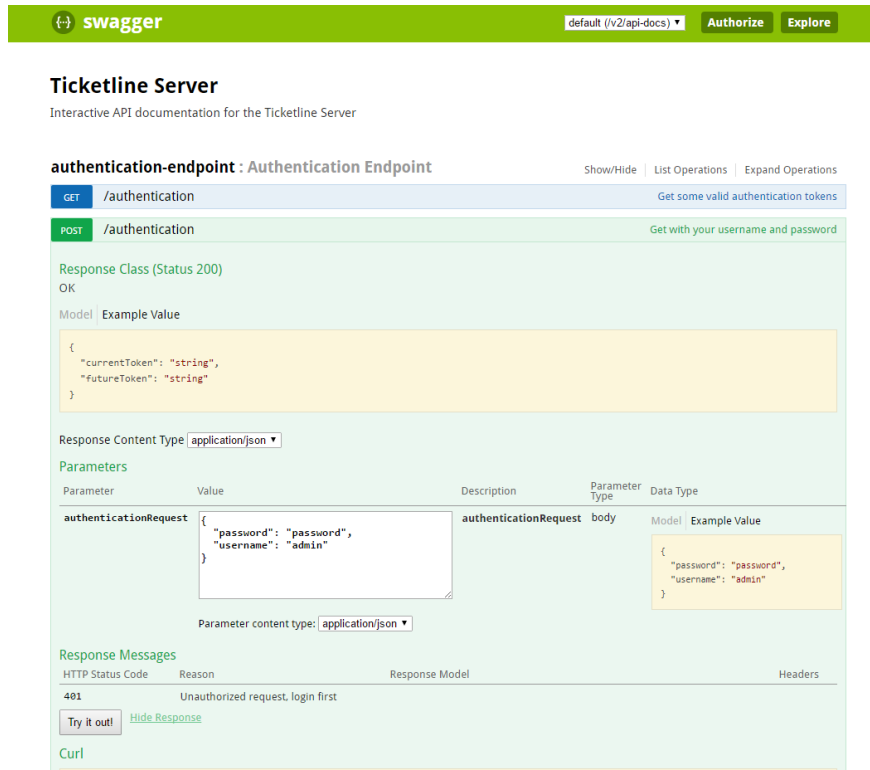
⁸<https://github.com/swagger-api/swagger-core/wiki/Annotations>

Listing 6: Swagger Annotations zur Dokumentation der REST-Schnittstelle

```
1 @Api(value = "news", description = "News_REST_service")
2 @RestController
3 @RequestMapping(value = "/news")
4 public class NewsController {
5
6     //...
7
8     @ApiOperation(value = "Gets_the_news_by_id", response = NewsDto.
9         class)
10    @RequestMapping(value =("/{id})", method = RequestMethod.GET,
11        produces = MediaTypeUtils.APPLICATION_JSON_VALUE)
12    public NewsDto getNewsById(@ApiParam(name = "id", value = "ID_of_
13        the_news") @PathVariable("id") Integer id) throws
14        ServiceException {
15        // ...
16    }
17
18    // ...
19
20 }
```


Das Swagger Projekt bietet mit SwaggerUI auch eine UI, die frei von Abhängigkeiten Ihre REST-Schnittstelle dokumentiert und leicht test- und bedienbar macht. Unter der URL <http://localhost:8080/swagger-ui.html> erreichen Sie, sobald der Server gestartet ist, die SwaggerUI (Siehe: Abbildung 4).

Achten Sie darauf, dass Sie sich gegenüber der REST-Schnittstelle authentifizieren müssen, da diese sonst für die meisten Anfragen eine “401 - Unauthorized“ Fehlermeldung liefert. Klicken Sie dazu in der SwaggerUI zuerst auf das Post-Request des authentication-Endpoints (siehe Abbildung 4) und schicken Sie Benutzernamen und Passwort an den Server, als Antwort erhalten Sie zwei Tokens (siehe Abbildung 5). Klicken Sie dann auf den Button “Authorize“ (siehe Abbildung 6), und fügen Sie hier den Token mit einem vorangestellten “Bearer“ ein. Danach sind Sie, je nach Rolle für die weiteren Anfragen Autorisiert. Der ausgestellte Token ist in der Standardkonfiguration 600 Sekunden lang gültig.



The image shows the SwaggerUI interface for the Ticketline Server. At the top, there's a green header with the Swagger logo, a dropdown menu set to 'default (/v2/api-docs)', and buttons for 'Authorize' and 'Explore'. Below this, the title 'Ticketline Server' is followed by the subtitle 'Interactive API documentation for the Ticketline Server'. The main section is titled 'authentication-endpoint : Authentication Endpoint' with links for 'Show/Hide', 'List Operations', and 'Expand Operations'. There are two tabs for the '/authentication' endpoint: 'GET' (blue) and 'POST' (green). The 'POST' tab is selected, showing a description 'Get with your username and password'. Below this, it indicates 'Response Class (Status 200)' is 'OK'. A 'Model' tab is active, showing an 'Example Value' as a JSON object: { "currentToken": "string", "futureToken": "string" }. The 'Response Content Type' is set to 'application/json'. Under 'Parameters', there's a table with columns: Parameter, Value, Description, Parameter Type, and Data Type. The 'authenticationRequest' parameter is shown with a value of { "password": "password", "username": "admin" } and a description of 'authenticationRequest body'. A 'Parameter content type' dropdown is set to 'application/json'. Below the parameters, 'Response Messages' are listed, showing a 401 status code with the reason 'Unauthorized request, login first'. At the bottom, there's a 'Curl' section.

Abbildung 4: SwaggerUI Authentication Endpoint



The image shows the SwaggerUI interface for the Token Response. It has two sections: 'Response Body' and 'Response Code'. The 'Response Body' section shows a JSON object with 'currentToken' and 'futureToken' values, both being long alphanumeric strings. The 'Response Code' section shows the status code '200'.

Abbildung 5: SwaggerUI Token Response

4.1.3 Service Package

Im Package *at.ac.tuwien.inso.sepm.ticketline.server.service* werden die Interfaces für die Service-Klassen definiert und das Sub-Package *implementation* enthält dementsprechend die Implementierungen dieser Interfaces (siehe Listing 7).

Listing 7: Spring Annotationen im NewsService

```
1 @Service
2 public class SimpleNewsService implements NewsService {
3
4     private final NewsRepository newsRepository;
5
6     public SimpleNewsService(NewsRepository newsRepository) {
7         this.newsRepository = newsRepository;
8     }
9
10    @Override
11    public List<News> findAll() {
12        return newsRepository.findAll();
13    }
14
15    // ...
16
17 }
```

Jede Service-Implementierung muss mit *@Service* annotiert sein. Andernfalls wird die Klasse nicht beim Starten des Servers instanziiert und das Injecten im Controller würde nicht funktionieren.

Auch im Service werden die Abhängigkeiten über den Konstruktor übergeben. Hierdurch wird die DAO, die für den Datenbankzugriff benötigt wird, injectet. Dies funktioniert nur,

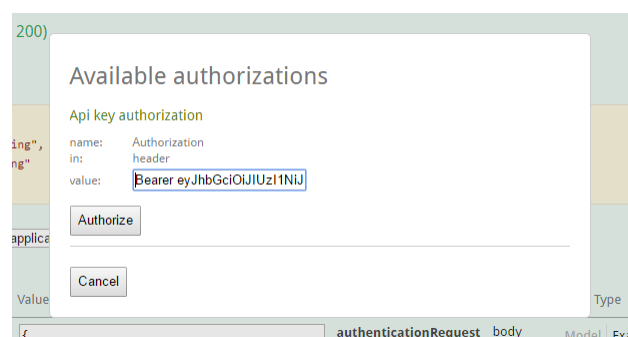


Abbildung 6: SwaggerUI Authorize Window

wenn die DAO Implementierung mit *@Repository* annotiert ist.

4.1.4 Entity Package

Dieses Package enthält die Entities für Ticketline. Die Klassen die für den Beispiel Anwendungsfall relevant sind, sind bereits enthalten.

Da in unterschiedlichen Ansichten jeweils nicht alle Daten benötigt werden, werden die Entity-Objekte mit Mappern in die entsprechenden REST-Objekte umgewandelt. Dazu wird die Library MapStruct⁹ verwendet. Mit ihrer Hilfe können sehr einfach automatisiert auch komplexere Mapper erstellt werden.

4.1.5 Repository Package

Das Package *at.ac.tuwien.inso.sepm.ticketline.server.repository* enthält bisher die Klasse *NewsRepository*, die Klasse erweitern *JpaRepository*. *JpaRepository* ist eine vom Spring Framework bereitgestellte Klasse, die CRUD-Funktionalitäten¹⁰ 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. Kompliziertere Abfragen lassen sich mittels JPQL und der *@Query* Annotation definieren. Weitere Informationen zu beiden Varianten finden Sie in der JPA Repository-Dokumentation¹¹.

4.2 Ticketline Client

In diesem Kapitel wird das Client-Projekt und dessen Packages genauer beschrieben, sowie deren Struktur und Verwendung erläutert. Abbildung 7 zeigt alle Packages des Clients und deren Abhängigkeiten.

Die Client Anwendung selbst ist wie der Server als SpringBootApplication erstellt.

⁹<http://mapstruct.org/documentation/1.1/reference/html/index.html>

¹⁰<http://de.wikipedia.org/wiki/CRUD>

¹¹<http://docs.spring.io/spring-data/jpa/docs/current/reference/html/>

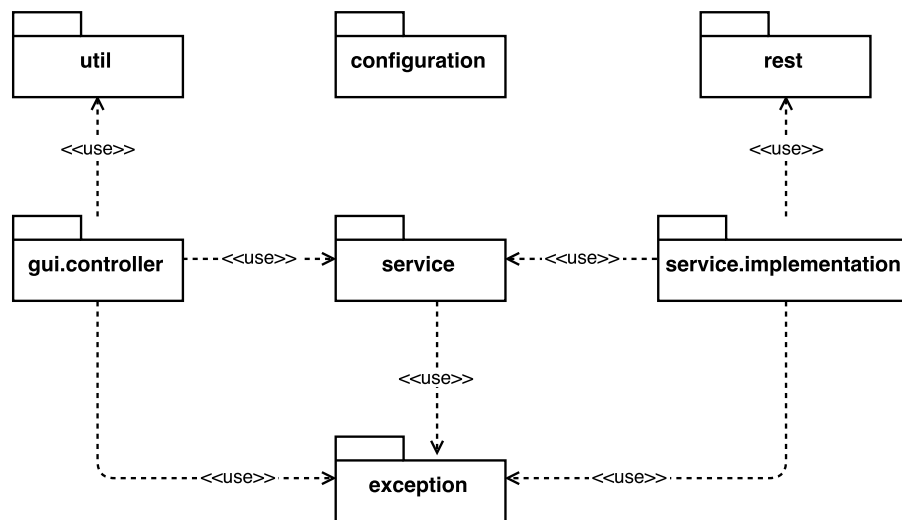


Abbildung 7: Ticketline Client Packages

4.2.1 Controller Package

Im Package *at.ac.tuwien.inso.sepm.ticketline.client.gui.controller* sind die GUI-Controller enthalten. Im Projekt sind bereits der *AuthenticationController*, der *NewsController* und der *NewsElementController* vorhanden. Die Controller müssen in den FXML-Dateien angegeben werden, die sich in *src/main/resources/fxml* befinden. Mit der *@FXML*-Annotation kann auf Felder, die in der FXML-Datei definiert wurden, zugegriffen werden (siehe Listing 8).

Die Controller selbst müssen mit der *@Component*-Annotation versehen sein, damit Spring Sie finden kann.

Listing 8: @FXML Annotation

```
1 @Component
2 public class AuthenticationController {
3
4     @FXML
5     private TextField tfUsername;
6
7     // ...
8
9 }
```

4.2.2 Service Package

Im Package *at.ac.tuwien.inso.sepm.ticketline.client.service* werden die Interfaces für die Services definiert. Die Implementierungen sind im Sub-Package *implementation* zu finden. Da der Client in den meisten Fällen sehr “dumm“ agiert wird hier in vielen Fällen nur validiert und an den Server weitergereicht.

4.2.3 REST Package

Im Package *at.ac.tuwien.inso.sepm.ticketline.client.rest* werden die Interfaces für die Kommunikation mit dem Server definiert. Die Implementierungen sind im Sub-Package *implementation* zu finden. Es ist bereits die komplette Authentication, sowie das Abrufen der News implementiert. Die URLs für die Verbindung zum Server werden aus der *application.yml*-Datei ausgelesen. Diese befindet sich in `src\main\resources\` hier werden wie schon beim Server alle gängigen Konfigurationswerte gespeichert.

4.2.4 Util Package

Das Package *at.ac.tuwien.inso.sepm.ticketline.client.util* enthält zum Beispiel 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. Wenn eine FXML-Datei geladen werden soll, muss dies immer über den *SpringFXMLLoader* geschehen, da nur dadurch die Spring-Beans mittels Dependency Injection im jeweiligen Controller gesetzt werden. Listing 9 zeigt die Verwendung des *SpringFXMLLoader*.

Listing 9: Verwendung des SpringFXMLLoader im MainController

```
1 @Component
2 public class MainController {
3
4     //...
5     private final SpringFXMLLoader springFXMLLoader;
6     //...
7
8     public MainController(SpringFXMLLoader springFXMLLoader,
9                           FontAwesome fontAwesome, AuthenticationInformationService
10                          authenticationInformationService) {
11         this.springFXMLLoader = springFXMLLoader;
12         //...
13     }
14
15     @FXML
16     private void initialize() {
17         //...
18         this.login = (Node) this.springFXMLLoader.load("/fxml/
19                 authenticationComponent.fxml");
20         this.spMainContent.getChildren().add(this.login);
21         //...
22     }
23 }
```

4.3 Ticketline REST

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.

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 UseCase, 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 <https://stackoverflow.com/>) nach Lösungen für Ihr Problem. Sollten Sie danach nicht weiterkommen, wird Ihr Tutor Ihnen beratend zur Seite stehen.