Cheftrainer – Football Manager

# Beschreibung des Spiels

# Anleitung

# Verwendete Komponenten

## Java FX

JavaFX ist ein plattformübergreifendes Framework zur Erstellung von graphischen Oberflächen. Wir verwendeten JavaFX für die Nutzeroberfläche der ClientApplication. Dieses Framework wurde verwendet, weil es unsere Meinung nach eine schöne Alternative zu Swing ist. Der Hauptgrund für den Einsatz von JavaFX ist jedoch die Tatsache, dass wir uns die Handhabung dieses Frameworks aneignen wollten.

## Log4J

Zum Loggen der Kommunikation zwischen Server und Client haben wir das Framework Log4j verwendet. Log4j ist Betriebssystemunabhängig, da es ausschließlich in Java zum Einsatz kommt. Log4j bietet die Möglichkeit an jeder Stelle des Codes einfach in ein Logfile zu schreiben. Zusätzlich sichert Log4j, dass das Logfile nicht unendlich weiter wächst, da Log4j mit einer maximalen Dateigröße konfiguriert werden kann.

## Maven

Das Build-Management-Tool Maven wurde in allen drei Subprojekten verwendet. Maven ist ein Tool, welches für den Erstellungsprozess einer Anwendung zuständig ist. Die einfache Einbindung von Bibliotheken war ausschlaggebend für den Einsatz in diesem Projekt. Durch dieses Dependency Management war es möglich, die ConnectorLib Plattform- und DIE-unabhängig in die ServerApplication und ClientApplication einzubinden. Die ConnectorLib, welche in einem eigenen Git-Repository liegt, kann direkt dorthin deployed werden.

## SQLite

SQLite ist eine Bibliothek, welche Datenbankfunktionen mit sich bringt. Diese Bibliothek benötigt keine weiteren Serverbedingungen, benötigt einen kleinen Speicherbereich und verwendet ein Art reduzierte SQL database engine. Durch diese Eigenschaften ist SQLite gut für das Projekt Cheftrainer geeignet, da das besagte Projekt ein vergleichsweise wenig umfangreiches ist und nur die grundlegendsten Datenbankfunktionen benötigt. Sie kommt ausschließlich in der ServerApplication zu Einsatz.

## JUnit

Für das automatisierte Testen der drei Subprojekte kam das Framework JUnit zum Einsatz. JUnit vereinfacht es Fehler während der Entwicklung zu finden oder gar zu vermeiden, da es mit jedem Erstellungsprozess der Anwendung besagte Tests ausführt und Rückmeldung über den Erfolg dieser liefert.

## JDOM

JDOM ist eine speziell für Java entwickelte Bibliothek, welche es einem ermöglicht XML-Dateien zu bearbeiten und zu lesen. Es wird in der ClientApplication benötigt um ein Menü in Abhängigkeit von einer XML-Datei zu generieren.

## JSON

Zum Datenaustausch zwischen Client und Server kommt JSON in diesem Projekt zum Einsatz. JSON ist ein einfach lesbares, einfach zu verendendes und kompaktes Datenformat. Diese Eigenschaften, gepaart mit der guten Erfahrung aus mehreren Projekten, sind der Grund für die Verwendung dieses Datenformats.

## Jsoup

Das Parsen von Websites, für die Automatisierung des Servers, wurde mithilfe von Jsoup realisiert. Jsoup ist eine Bibliothek, welche das Lesen und Bearbeiten von HTML-Dateien ermöglicht. Es ist deutlich komfortabler und einfacher HTML mittels Jsoup zu parsen als auf einen reinen XML Parser zurückzugreifen.

# Aufbau der Anwendung

Das gesamte Projekt ist in 3 Subprojekte aufgeteilt, die ServerApplication, die ClientApplication und die ConnectorLib. Diese Aufteilung ermöglicht es die ClientApplication und die ServerApplication voneinander unabhängig zu entwickeln. Beide Projekte inkludieren die ConnectorLib. Diese stellt wichtige Komponenten bereit, welche von beiden Projekten benötigt werden. Dazu zählen das Modell, sowie die Kommunikation zwischen der ClientApplication und der ServerApplication.

## ServerApplication

Die Anwendung, welche auf dem Server läuft. Sie kommuniziert mit einer oder mehreren ClientApplications. Zusätzlich verwaltet sie die SQLite Datenbank und regelt den Zugriff auf diese. Außerdem erledigt sie autonome Aufgaben, wie das Erzeugen der Datenbank, beim ersten Start der Anwendung, anhand der Website [www.sportal.de](http://www.sportal.de). Dazu wird das HTML der Website mittels JSoup geparsed. Während des laufenden Betriebs werden weitere autonome Aktionen, wie das „updaten“ der Spielerpunkte, nach einem Spieltag, oder das Ausführen von Transaktionen.

## ClientApplication

Die ClientApplication ist im Wesentlichen für die Interaktion mit dem Server zuständig. Außerdem erzeugt sie die GUI, welche mittels Java FX realisiert wurde, mit welcher der Nutzer interagieren kann.

## ConnectorLib

Die ConnectorLib beinhaltete das Modell der Anwendung. Zusätzlich bietet sie die Möglichkeit, eine Server/Client Anwendung leicht zu erstellen. Dadurch erhält die ClientApplication einen MessageController und die ServerApplication einen MessageController pro eingeloggten Client. Dem MessageController können Message Objekte übergeben werde. Diese werden automatisch in JSON Formatiert und AES verschlüsselt übertragen. Damit die AES Verschlüsselung korrekt funktioniert führt die ConnectorLib direkt nach der Verbindung einen Handshake zwischen Client und Server durch. Dabei wird ein AES Key erzeugt und RSA verschlüsselt zwischen Client und Server übertragen.

Ein Message Objekt besteht im Wesentlichen aus einer ID und der eigentlichen Nachricht, welche als String übergeben werden kann. Die ConnectorLib stellt zwei Klassen, eine für „Client-To-Server“ und eine für „Server-To-Client“ Message IDs zur Verfügung. Die dort registrierten IDs können einer Message im Konstruktor übergeben werden. Bei der Erstellung des Message Controllers werden diese IDs auf Klassen gemapped. Dafür wird ein Classloader verwendet, welcher Instancen der Klassen erzeugt und diese in einer Map speichert. Diese Klassen müssen in einem „callables“ Package liegen, der Name der Klasse muss äquivalent zu dem Wert der ID sein und sie müssen von CallableAbstract erben. Dies hat den Vorteil, dass kein Switch-Case verwendet werden muss. Eine Nachricht wird direkt „messageArrived(Message message)“ der Klasse mit der entsprechenden ID weitergeleitet.

# Visualisierung

# Beschreibung des Umsetzungsgrades

# Erkenntnisgewinn