Projekt: Trading Service

Projektbeschreibung

Das Ziel des Projekts ist der Entwurf einer Architektur für ein verteiltes System und die Einarbeitung in Middleware-Technologien für verteilte Systeme basierend auf der Java Enterprise Edition. Zu diesem Zweck werden Sie eine einfache Bankanwendung implementieren, die ein TradingService zum Kauf und Verkauf von Aktien für Kunden aufruft. Weiters werden Sie Client-Anwendungen implementieren, die Mitarbeitern bzw. Kunden zur Verfügung gestellt werden, um Online-Transaktionen durchzuführen.

Anforderungen

Das Projekt teilt sich in eine Server- und Client-Implementierung auf. Für den **Bankserver** (die Bank) ergeben sich folgende Aufgaben:

- Die Bank ermöglicht es Kunden, Anteile von verfügbaren Aktien zu suchen, kaufen und verkaufen.
- Die Bank verwaltet Depots für Kunden und speichert für jeden Kunden mit, wieviele Anteile für welche Aktien sich in seinem Depot befinden.
- Um Aktien für einen Kunden zu kaufen oder zu verkaufen, verwendet die Bank ein Web Service der Börse. Die WSDL dazu ist unter folgendem Link zu finden: http://edu.dedisys.org/ds-finance/ws/TradingService?wsdl. Zwecks Überbrückung der Zeit bis zur WS-Einheit gibt es auch eine Javadoc-Beschreibung der Schnittstelle. Der Aufruf des Web Services erfordert eine Authentifizierung. Die Authentifizierungsdaten erhalten Sie bei der Projekteinweisung.
  Die Finanzdaten ab 24. November 2017 werden von IEX zur kostenfrei zur Verfügung gestellt und dürfen nur im Rahmen dieser Lehrveranstaltung verwendet werden. Die IEX-Bedingungen zur Datennutzung finden sich unter https://iextrading.com/api-exhibit-a.
- Das investierbare Volumen an der Börse ist pro Bank initial auf eine Milliarde Dollar beschränkt. Der Kauf bzw. Verkauf von Aktien verringert bzw. erhöht diesen Wert entsprechend. Die Bank führt Buch über diese Summe und passt sie nach an der Börse getätigten Käufen bzw. Verkäufen entsprechend an.
  Sie können die Investment-Performance (aktuelles Gesamtvolumen) Ihrer Bank im Vergleich zu den Banken der anderen Gruppen auf folgender Seite einsehen: http://edu.dedisys.org/ds-finance/
- Der Bankserver bietet remote Schnittstellen für Clients an, mit denen Kunden bzw. Mitarbeiter Transaktionen online durchführen können.

Mittels **Client für Mitarbeiter** können Mitarbeiter der Bank zumindest folgende Aktionen durchführen:

- Anlegen von Kunden, wobei für einen Kunden mindestens Vor- und Nachname, Adresse und eine Kundennummer vergeben werden.
- Suchen nach Kunden mittels Kundennummer oder Name des Kunden. Gehen Sie dabei davon aus, dass Kunden nicht immer ihre Kundennummer wissen.
- Suchen nach verfügbaren Aktien.
- Kaufen von Aktien für einen Kunden.

- Verkaufen von Aktien für einen Kunden.
- Auflisten aller Aktienanteile im Depot eines Kunden inkl. aktuellem Wert pro Firma und Gesamtwert des Depots.
- Abfrage des aktuell investierbaren Volumens der Bank an der Börse.

Der **Client für Bankkunden** ermöglicht einem Kunden die Verwendung zumindest folgender Funktionen:

- Suchen nach verfügbaren Aktien.
- Kaufen von Aktien
- Verkaufen von Aktien
- Auflisten aller Aktienanteile im Depot inkl. aktuellem Wert pro Firma und Gesamtwert des Depots
- Um seine Aufgaben erfüllen zu können, verwendet der Bankclient das zur Verfügung gestellte Interface des Bankservers.

Überlegen Sie sich, wie Sie die von Kunden und Mitarbeitern gemeinsam verwendete Funktionalität sinnvoll nur einmal programmieren müssen, ohne dass ein Kunde in der Lage ist, für einen anderen Kunden Transaktionen durchzuführen. Die Annahme dabei ist, dass ein technisch versierter Kunde mit Programmierkenntnissen durchaus auch seinen eigenen Client schreiben könnte, wenn er das Interface des Bankservers kennt.

Gruppen mit weniger als fünf Personen können den Client für Bankkunden in der Implementierung weglassen. Für den Design-Teil überlegen Sie sich aber bitte die Aufgabenstellung hinsichtlich einmaliger Implementierung gemeinsamer Funktionalität unter entsprechenden Sicherheitsvorkehrungen.

Technologie

Zur Realisierung der Übungsaufgabe verwenden wir die Java Enterprise Edition mit Wildfly Application Server 10.1.0. Programmiersprache ist daher Java.

Bewertung

Die Bewertung basiert auf folgenden Teilen:

- 30% Projektdesign (Abgabe: Projekt Teil 1)
- 30% Server Realisierung
- 30% Client Realisierung
- 10% Präsentation

Es wird dabei jeweils darauf geachtet, inwieweit die obigen Anforderungen erfüllt sind.

Sie erhalten nach Projekt Teil 3 pro Gruppe eine Gesamtpunktesumme, die sich aus der Gruppenleistung nach obiger Bewertung multipliziert mit der Anzahl der Gruppenmitglieder ergibt, z.B. 85 * 6 = 510. Diese Punktesumme können Sie in der Gruppe untereinander verteilen, z.B. 2*95 + 2*85 + 2*75 = 510. Diese Verteilung schickt dann der Gruppensprecher an den jeweiligen Lektor mit Kopie an alle Gruppenmitglieder. Sollten Sie sich nicht einigen können, dann wird die Punkteverteilung durch den Lektor vorgenommen.

Sollten Sie beim Abgabetermin keine funktionierende (positive) Lösung präsentieren können, verfallen die Präsentationspunkte. Sie können dann eine positive Lösung nachbringen, wobei dies bereits als 1. Wiederholung des Projekts zählt.

Aufgaben

Sollten Sie Fragen oder Probleme beim Lösen der Aufgaben haben, sehen Sie bitte zuerst nach, ob diese nicht bereits in den Hinweisen und FAQs zum Projekt beschrieben sind.

Projekt - Teil 1

Bis zur ersten Übung bereiten Sie bitte das Design für Ihre Gruppe vor, um es in der Übungseinheit präsentieren und diskutieren zu können. Für Präsenation und Diskussion stehen insgesamt ca. 30 Minuten pro Gruppe zur Verfügung. Damit sollte eine Gruppen-Präsentation insgesamt nicht länger als 20 Minuten dauern, um eine anschließende Diskussion zu ermöglichen.

Konkret soll das Design bzw. die Präsentation folgende Punkte beinhalten:

1. Architektur des Gesamtsystems inkl. Spezifikation der Funktionen, die von den einzelnen Teilen zur Verfügung gestellt werden. Ordnen Sie dabei die von Ihnen zu erstellenden Artefakte in einem Architekturdiagramm der Java Enterprise Architektur zu.
2. Design des Mitarbeiter-Clients inkl. Interaktionen mit dem Server
3. Design des Kunden-Clients inkl. Interaktionen mit dem Server
4. Spezifikation der am Bankserver persistent verwalteten Daten (JPA Entities).
5. Remote Interface-Definition des Bankservers (inkl. typisierten Methodenparametern) und Festlegung der Technologie (RMI oder Web Services).

Weiters führen Sie bis zu diesem Termin bitte folgende Schritte durch, um etwaige Setup-Probleme frühzeitig zu erkennen:

1. Importieren des Projekttemplates aus dem Subversion Repository in die Entwicklungsumgebung.
2. Deployment des Enterprise Archivs aus dem Projekttemplate im Wildfly Application Server.

Abgabe: Abzugeben ist *vor* dem Präsenztermin eine kurze Ausarbeitung (PDF, DOC, PPT) mit der Beschreibung des Designs für Ihre Gruppe als eine ZIP Datei per E-Mail an den jeweiligen Lektor – mit CC an alle Gruppenmitglieder. Die Datei soll dabei nach folgendem Schema benannt sein: Gruppenkürzel_NachnameGruppensprecher_Trading_Teil1.zip, also z.B. 05_Maier_Trading_Teil1.zip. Es ist ausreichend, die bei der Präsentation verwendete Powerpoint-Präsentation abzugeben, sofern diese alle geforderten Elemente enthält.

Projekt - Teil 2

Teil zwei dient dem frühzeitigen Feststellen von grundsätzlichen Problemen mit dem Umgang der Technologie. Dazu bereiten Sie bitte bis zur zweiten Einheit folgende Punkte vor:

1. Mündlicher Bericht über aktuellen Zwischenstand und eventuelle Probleme.
2. Prinzipielle Durchführung eines Serveraufrufs mittels Client (Kunden- oder Mitarbeiterclient).
3. Aufruf zumindest einer Web Service Operation am TradingService.
4. Persistierung einer JPA Entity.

Projekt - Teil 3

Fertige Implementierung und Vorführung in der Übung. Abgabegespräch mit Prüfungscharakter:

1. Kurzpräsentation des Kerns des Projekt-Designs (Architektur, Grundfunktionalität, Depotverwaltung, ...) und Zusammenfassung von etwaigen Änderungen gegenüber dem ursprünglichen Design.
2. Vorführung der Implementierung.
3. Erklärung spezifischer Mechanismen und Codeteile.

Im Rahmen des Abgabegesprächs geht es darum, Ihr Verständnis der Materie zu demonstrieren, d.h. einerseits Ihre Implementierung erklären zu können und andererseits die Artefakte Ihrer Implementierung auf die theoretischen Inhalte z.B. der Java Enterprise Architektur zuordnen zu können. Dies gilt für Gruppenmitglieder auch individuell, d.h. dass bei mangelndem Verständnis auch vereinzelte Zweitabgaben (zweiter Antritt zum Projekt) möglich sind.

Abgabe: Als Abgabe zählt jener Stand in Ihrem Subversion Repository, der als letzter *vor* dem Abgabetermin eingecheckt wurde.

Downloads

Software Setup:

1. Installieren Sie das Java Development Kit (JDK) JDK 8
2. Installieren Sie die NetBeans 8.2 in der Version "Java EE" oder "All". Achtung: NetBeans 8.2 ist mit JDK ≥ 9 nicht kompatibel.
3. Wildfly Application Server 10.1.0 downloaden und lokal entpacken.
4. Wildfly in NetBeans einrichten: Menü "Tools - Servers - Add Server..." Hinweis: Sollten Sie diesen Menüpunkt nicht sehen, liegt es vermutlich daran, dass in Ihrem NetBeans noch kein Projekt existiert.
5. *Optional*: Installieren Sie TortoiseSVN: Windows Explorer Plugin zur Verwaltung des Quellcodes mit grafischer Oberfläche außerhalb eine IDE. Hilfreich ist dieser Client z.B. bei der Behebung von Konflikten.

Entwicklungs-Framework: Das Framework für die Entwicklung wird Ihnen in einem Subversion Repository pro Gruppe zur Verfügung gestellt. URL und Zugangsdaten erhalten Sie bei der Projekteinweisung.

Hinweise zum Projektstart

1. **Generierung von Java Code für Web Services basierend auf WSDL:** Das zur Verfügung gestellte Framework ist Maven-Modul-mäßig analog zum EJB Calculator

Beispiel aus der Übung aufgebaut. Im EJB-Modul und im Client-Modul gibt es jeweils ein Verzeichnis src/main/resources/META-INF/wsdl-consumed. Der Build-Prozess ist so konfiguriert, dass für WSDL-Dateien, die Sie in dieses Verzeichnis kopieren, der zugehörige JAX-WS Java Code zur Verwendung des Web Services generiert wird(unter target/generated-sources). Die Dateien müssen dabei die Endung ".wsdl" haben.

2. **Wildfly Accounts für Remote Zugriff**Für den Remote-Zugriff des Clients müssen in Wildfly entsprechende Accounts registriert werden. Verwenden Sie zur Anlage von Kunden- bzw. Mitarbeiteraccounts das "add-user" Skript, das im JBOSS_HOME/bin Verzeichnis zu finden ist, wobei JBOSS_HOME für das Verzeichnis steht, in dem Sie Wildfly entpackt haben. Achten Sie darauf, dass Sie für Kunden und Mitarbeiter unterschiedliche Rollen vergeben. Abfragen können Sie diese dann in Ihrer Anwendung über den `SessionContext` (`isCallerInRole(role)`, siehe auch Studienbrief). Gespeichert werden diese Werte unter JBOSS_HOME/standalone/configuration in application-users.properties bzw. application-roles.properties.
3. **Utility Klasse zur Anlage von Wildfly Accounts innerhalb Ihres Programms:** Zwecks Projektvereinfachung ist es leider erforderlich, dass Sie Kunden sowohl in Ihrer Applikation verwalten (zwecks Suche) als auch in Wildfly anlegen (Authentifizierung). Für Wildfly kann dies über das add-user Skript erfolgen. Zur Anlage von Wildfly-Accounts steht Ihnen aber auch eine Utility-Klasse im EJB Modul des Frameworks zur Verfügung. Der Name des Users bzw. dessen Rolle können so wie zur Authentifizierung verwendet über den `SessionContext` abgefragt werden.
4. **Web Service Konfiguration für Server:** Sollten Sie die Kommunikation zwischen Client und Server mittels Web Services implementieren, so müssen Sie zur Aktivierung der Authentifizierung eine entsprechende Konfiguration in der Datei src/main/resources/META-INF/jboss-webservices.xml vornehmen. Eine Beispielkonfiguration ist dort bereits vorhanden.
5. **Datasource Konfiguration:** Die JPA Konfiguration ist so eingestellt, dass Sie nur noch auf der `EntityManager` API zum Laden und Speichern von Entities aufsetzen müssen. Allerdings ist noch die Konfiguration einer Datasource im Container erforderlich. Dazu gibt es zwei Möglichkeiten:
    - *Variante 1*: Konfiguration mittels Administrationsoberfläche:
        1. Fügen Sie mittels "add-user" Skript im bin Verzeichnis von Wildfly einen Management User zum Wildfly hinzu.
        2. Starten Sie Wildfly und rufen im Browser folgende URL auf: http://localhost:9990/
        3. Wählen Sie Configuration → Subsystems → Datasources → Non-XA → Add
        4. Wählen Sie die "H2 Database" mit folgendenden Daten:
            - Name: DsFinanceBankDS
            - JNDI Name: java:/datasources/DsFinanceBankDS
        5. Beim Treiber wählen Sie unter "Detected Drivers" den existierenden Treiber "h2" aus. Dieser ist im Wildfly schon standardmäßig konfiguriert.
        6. Spezifizieren Sie folgende Connection Settings:
            1. Connection URL: jdbc:h2:~/ds-finance-bank
            2. Username: sa
            3. Password: sa
    - *Variante 2*: Direkte Konfiguration in der Konfigurationsdatei:
      In dem Fall ist folgender Eintrag im `<datasources>` Abschnitt von

JBOSS_HOME/standalone/configuration/standalone.xml erforderlich – abhängig davon, was bei Ihnen in in NetBeans in der Serverkonfiguration konfiguriert ist (Tools – Servers), müssen Sie ggf. die Änderungen in standalone-full.xml durchführen:

```
<datasource jndi-name="java:/datasources/DsFinanceBankDS"
    pool-name="DsFinanceBankDS" enabled="true" use-java-context="true"
    use-ccm="true">
  <connection-url>jdbc:h2:~/ds-finance-bank</connection-url>
  <driver>h2</driver>
  <security>
      <user-name>sa</user-name>
      <password>sa</password>
  </security>
</datasource>
```

Eingefügt werden muss diese Datasource Konfiguration zwischen folgenden zwei Zeilen:

```
</datasource>
  <drivers>
```

6. Durch diese Konfiguration erzeugt Wildfly beim Start eine Datei namens "ds-finance-bank.h2.db" in Ihrem Home-Verzeichnis. Wenn Sie diese Datei gerne woanders speichern möchten, müssen Sie den String "~/ds-finance-bank" entsprechend anpassen. Hintergrund: Diese Datei wird von der in Wildfly integrierten Java-Datenbank H2 erstellt. Sollten Sie sich den Inhalt der Datenbank ansehen wollen, so ist das z.B. wie folgt möglich:
    1. Installieren Sie den Squirrel SQL Client
    2. Registrieren Sie den JDBC Treiber für H2 in Squirrel SQL unter "Drivers - H2 - Modify Driver ... - Extra Class Path - Add" und wählen Sie dort die h2-*.jar Datei aus dem Verzeichnis WILDFLY_HOME\modules\system\layers\base\com\h2database\h2\main aus. Wenn Sie die Registrierung korrekt abgeschlossen haben, erscheint unter Drivers bei "H2" ein blaues Hakerl statt dem roten "x".
    3. Konfigurieren Sie die Datasource in Squirrel SQL mittels "Aliases - Add Alias". Im erscheinenden Dialogfenster vergeben Sie einen Namen, wählen "H2" als "Driver" aus und geben URL, Benutzername und Passwort genau so ein, wie oben in der Datasource Konfiguration.
    4. Mittels "Test" können Sie prüfen, ob einen Verbindung hergestellt werden kann.
    5. Nach Abschluss durch "OK" können Sie durch Doppelklick auf den soeben erstellten Alias eine Verbindung zur Datenbank herstellen. Diese wird allerdings leer sein, solange Sie noch keine persistenten Entitäten mittels JPA definiert bzw. gespeichert haben.

Beachten Sie bitte, dass die Datasource so konfiguriert ist, dass die Datenbankverbindung per Dateizugriff erfolgt. Damit ist kein eigener Datenbankserver erforderlich, es kann aber nicht gleichzeitig von zwei Prozessen auf dieselbe Datei zugegriffen werden. Damit funktioniert der Zugriff per Squirrel SQL nur dann, wenn Wildfly gestoppt ist (und umgekehrt).

Hinweise für Interessierte

1. Im zur Verfügung gestellen Framework ist die SLF4J Logging API eingebunden. Siehe SLF4J Manual bezüglich Verwendung. Für die Konfiguration muss zwischen Client und Server unterschieden werden:
    - Client: Am Client wird die Logback Implementierung der SLF4J API verwendet. Die Konfigurationsdatei dazu ist unter `src/main/resources` im Client-Projekt zu finden.
    - Server. Am Server ist die Logging Konfiguration in JBOSS_HOME/standalone/configuration/standalone.xml zu finden. Damit Sie am Server auch Logger.debug Messages auf der Konsole ausgegeben bekommen, müssen Sie den folgenden Eintrag:
    - ```
      <console-handler name="CONSOLE">
            <level name="INFO"/>
      ```

      wie folgt ändern:

      ```
      <console-handler name="CONSOLE">
                <level name="DEBUG"/>
      ```
      ```