Verteilte Systeme- API Projekt

Namo Abdulla 10013241

Tim Boddenberg 10011167

Inhalt

Installation und sonstige wichtige Start Hinweise 1
Konkrete Schritt für Schritt Anleitung 1
Datenbank 2
Maven Repository 2
Unit Tests 2
Swagger UI 2
Rest API-Design 2
Programmiersprache und Framework 2
Teststrategie 3
Manuelles Testing 3
Swagger 3
Unit-Tests 3
Limitierungen und Hinweise 4

Installation und sonstige wichtige Start Hinweise

Das Projekt hat ein paar Abhängigkeit, dessen Korrektheit wichtig ist, um die Lauffähigkeit des Projektes zu garantieren.

Konkrete Schritt für Schritt Anleitung

Diese Anleitung ist etwas oberflächlicher, genauere Hinweise zu den Punkten werden nachfolgend sehr detailliert beschrieben.

1. Projekt klonen
2. Datenbank anlegen
    a. In der Konfiguration ist diese unter dem Port 3306 zu finden, sollte das bei Ihnen anders sein, steht eine detailliertere Anleitung unter dem Punkt „Datenbank“.
    b. Es muss eine mindestens leere Datenbank mit dem Namen „verteiltesysteme“ angelegt werden.
    c. Wir haben dafür Xampp genutzt, dort den Apachen und eine MySQL Instanz gestartet
    d. Nutzername und Passwort können ebenfalls in der Konfiguration angegeben werden (siehe Punkt „Datenbank“)
3. Applikation über die Klasse „VerteilteSystemeApplication“ starten

   a. ACHTUNG: Wenn die Datenbank noch nicht existiert, startet die Application nicht.
4. Für das Starten der Unit Tests bitte die detaillierte Anleitung unter „Unit Tests“ lesen.

Datenbank

Als Erstes ist es wichtig, dass eine Datenbank existiert. Beim Start des Projektes wird diese direkt angefragt, bei einem Verbindungsfehler wird der Startvorgang abgebrochen. Es muss hier über XAMPP eine Apache Instanz gestartet werden, um darüber einen lokalen MySQL Server zu hosten. Auf diesem muss eine Datenbank mit dem Namen „verteiltesysteme“ vorliegen, diese kann vollkommen leer sein. Solange es sich dabei um eine MySQL Datenbank handelt, kann diese auch auf andere Art und Weise gehostet werden. Sollten dabei die Verbindungsparameter von den nachfolgenden abweichen, müssen diese unter „resources/application.properties/“ abgeändert werden.

```
spring.datasource.url=jdbc:mysql://localhost:3306/verteiltesysteme
spring.datasource.username=root
spring.datasource.password=
```

Maven Repository

Als Nächstes sei erwähnt, dass in dem Projekt Maven Abhängigkeiten existieren. Die Dependencies werden im File „pom.xml“ festgehalten. Normalerweise sollte die IDE sich automatisch diese Abhängigkeiten ziehen, sollte dies nicht der Fall sein, müsste der Download manuell angestoßen werden. Wie bereits erwähnt sollte jede vernünftige IDE diesen Schritt von allein übernehmen und somit kein manuelles Anstoßen der Synchronisation nötig sein.

Unit Tests

Die Unit Tests decken über drei Tests jeden Controller ab. Wenn man über die IDE die Tests in der jeweiligen Testklasse startet, wird die Reihenfolge verändert und hat somit keinen Mehrwert mehr. Die richtige Reihenfolge der Tests ist AddAndGet (POST,GET), Change (PUT) und als letztes Delete (DELETE). Die Tests müssen also einzeln in der angegebenen Reihenfolge gestartet werden, da sonst der PUT-Befehl nach den DELETE-Befehl ausgeführt wird.

Swagger UI

Die Dokumentation über Swagger erzeugt nicht nur eine XML/Json, sondern verfügt ebenfalls über ein UI. Sobald das Projekt gestartet ist, ist es unter Localhost:8080/swagger-ui.html zu erreichen. Dort können nicht nur Dokumentationen eingesehen werden, sondern auch die entsprechenden Schnittstellen getestet werden.

Rest API-Design

Für das Projekt wurde von uns das API-Design Nummer 3 ausgewählt. Nach dem Überprüfen der vier Designs kam uns dieses am logischsten und intuitivsten vor und wurde daher von uns als Vorlage genutzt und umgesetzt.

Programmiersprache und Framework

Auf die Programmiersprache konnte sich unsere Gruppe schnell einigen. Java kommt der Sprache C# sehr nah, in welcher wir uns beide deutlich besser auskennen. Da diese nicht zur Wahl stand, einigten wir uns auf Java, da bereits in den vorherigen Semestern mit Java gearbeitet wurde.

Beim Framework ist die Wahl ebenfalls schnell gefallen. Spring Boot bietet über ein Annotation Handling ein sehr übersichtliches Mapping der Requests an, deshalb war es sehr gut für eine API geeignet, die über Operations arbeitet. Entitäten und dessen Verwaltung sind ein sehr großer

Bestandteil der Anforderungen an das Projekt. Das Framework bietet eine sehr saubere Lösung, wie man genau dies umsetzen kann.

Teststrategie

Die API kann auf viele Arten getestet werden, auf welche nachfolgend eingegangen wird.

Manuelles Testing

Für diese Methode bietet sich die von uns genutzte Software Postman an. Über diese kann man jede API-Operation ganz einfach testen. POST bzw. PUT lassen sich über diese Software sehr einfach abbilden, da das benötigte Json Objekt einfach dem Header angehängt werden kann. Das genaue Vorgehen ist hierbei, dass man in die Controller schaut und die einzelnen Methoden abarbeitet. Jede Methode wird dann über die Software Postman abgefragt und das Ergebnis kontrolliert. So wurde die Software auch anfangs getestet.

Swagger

Das Swagger Interface bietet eine Übersicht über alle Methoden der API. Wenn man diese genauer betrachtet, gibt es einen Button „try it out“. Wenn man diesen klickt, kann man einen Request absenden und auch Parameter übergeben. Nachdem Swagger im Projekt installiert war, wurde diese Methode verwendet, um die Software nach dem Prinzip des manuellen Testings zu testen.

Unit-Tests

Für das automatische Testen der Applikation wurden Unit Tests implementiert. Diese funktionieren nach folgendem Prinzip:

AddAndGet

Damit der Eintrag mit Sicherheit angelegt wird, wird zuerst ein Get-Request gesendet und geprüft, ob ein Eintrag bereits vorliegt, der dem für den Test entspricht. Ist ein Eintrag vorhanden, wird dieser gelöscht. Anschließend wird ein Eintrag angelegt und erneut ein Get-Request auf diesen Eintrag gesendet um zu verifizieren, dass der Eintrag soeben angelegt wurde.

Change

Für diesen Test wird ein Get-Request auf den angelegten Eintrag aus dem „AddAndGet“-Test ausgeführt und in das entsprechende Objekt gecastet. Danach wird ein Attribut über einen Setter überschrieben, in einen Json-String überführt und als PUT-Request an den Controller gesendet. Danach wird erneut ein Get-Request auf diesen Eintrag gesendet und überprüft, ob das Attribut überschrieben wurde.

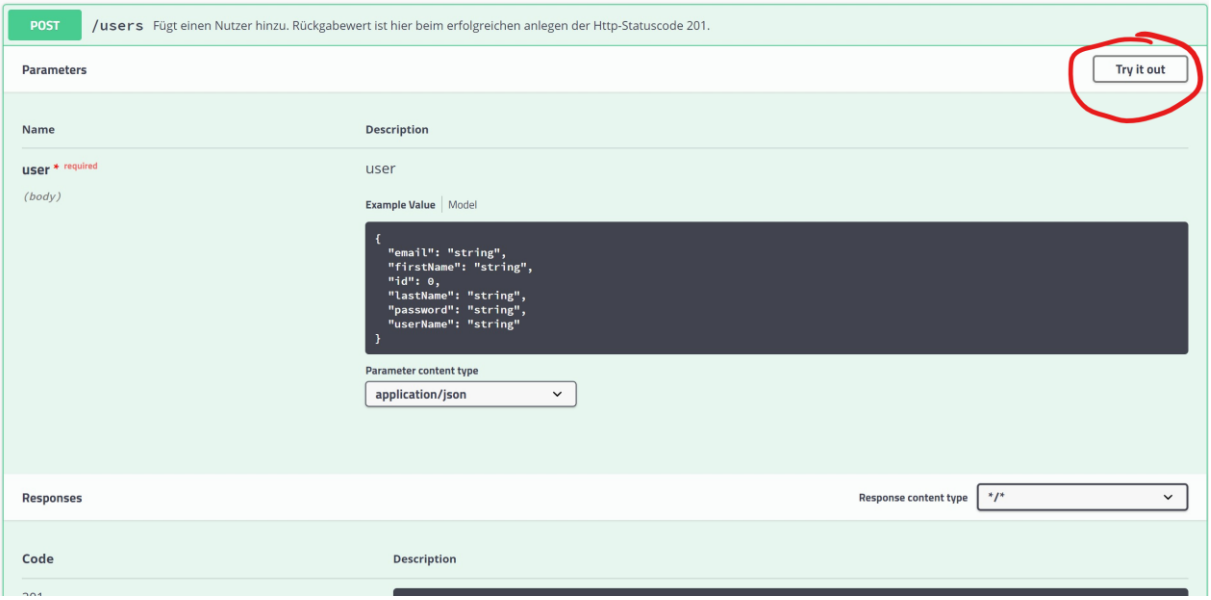
Delete

Bei diesem Test wird am Anfang wieder ein Get-Request gesendet, um sicherzugehen, dass der Eintrag wirklich vorliegt. Danach wird die Delete Operation auf diesen Eintrag ausgeführt und erneut ein Get-Request gestellt. Wenn zurückgegeben wird, dass der Eintrag nicht gefunden wurde, ist der Test erfolgreich.

Wie anfangs bei der Installation erwähnt, ist es wichtig, dass die Tests in der korrekten Reihenfolge ausgeführt werden. Wenn man die Tests automatisch über die IDE ausführen lässt, wird die Reihenfolge vertauscht und der letzte Test schlägt fehl. Es ist also wichtig, dass die Tests einzeln in der richtigen Reihenfolge ausgeführt werden.

Limitierungen und Hinweise

Im ausgewählten Pattern sind uns bei den beispielhaft übergebenen Json-Strings Inkonsistenzen aufgefallen. Zum Beispiel beim Bereich „POST User“ sind uns gleich mehrere davon aufgefallen, wenn diese per Copy Paste zum Testen an unsere Applikation übergeben werden, können diese nicht in das entsprechende Objekt gecastet werden, da sich dieses an der für uns richtigen Version orientiert (Camel Case, doppelte Anführungszeichen, …). Es ist also wichtig darauf hinzuweisen, dass die Json Strings, gerade beim POST oder PUT sich an der Schreibweise der Entities im Projekt orientieren. Am einfachsten ist es, das Swagger UI aufzurufen und dort unter „Try it out“ beim POST oder GET Menüpunkt das Schema abzuschauen und dieses an die Applikation zu senden.



Nach dem Klick auf „Try it out“ kommt ein Eingabefeld, in welchem das Schema bereits vorgegeben ist. Dort kann man die Values ersetzen und per „Execute“ an die Applikation senden. Wichtig ist hierbei die Unique-Felder zu beachten, da zum Beispiel ein doppelter Nutzername nicht hinzugefügt wird, gleiches gilt auch bei anderen Entities, daher vorher in die Entity im Quellcode schauen, ob Unique-Attribute vorliegen.

Des Weiteren ist uns aufgefallen, dass man bei einem PUT-Request die Id des zu überschreibenden Eintrages als Route-Parameter übergeben muss. Wir haben das ganze so gelöst, dass man eine Id übergeben muss, damit die Route aufgelöst wird, wie im Design gewünscht. Die entscheidende Id allerdings, die den Eintrag im Repository identifiziert ist die, die mit dem Json String übergeben wird. Anhand der Id des neuen User Objektes wird das bestehende überschrieben, nicht anhand des Route Parameters.