Anwendungssysteme (SS 2019)

Dr.-Ing. Anselm Busse, Dominik Ernst, Jonathan Heiß

Bewertete Programmieraufgabe 2 (10 Portfoliopunkte)

Im Rahmen dieser Programmieraufgabe erstellen Sie mit Ihrer Gruppe das Serverseitige Backend (d.h. Anwendungslogik und Datenmanagement) für eine Chatanwendung.

Aufgabenstellung

Ihre Aufgabe ist es das Server-seitige Interface („Backend“) der Chat-Anwendung *WhatsChat* zu implementieren. Das Backend soll dabei lokal auf Ihrem Rechner laufen. Das WhatsChat-Frontend entspricht dabei der von Ihnen in der ersten Programmieraufgabe entwickelten Komponente. Es gibt also erneut Gruppenchats (*Channels*), in die Nutzer beitreten können, um dort Nachrichten (*Messages*) auszutauschen. Das Backend soll eine http-basierte Programmierschnittstelle (API) zur Verfügung stellen. Eine Beschreibung des Soll-Zustands der API ist im Anhang dieses Dokuments zu finden.

Für die Umsetzung des Backends ist Java mit Spring Boot zu verwenden. *Eine Projektvorlage dazu finden Sie im ISIS-Kurs. Das Backend soll mit dem in Aufgabe 1 erstellten Frontend lauffähig sein (Serveradresse stattdessen jedoch „localhost“). Sofern Ihr Frontend unvollständig oder fehlerhaft war, können Sie als Referenz und zu Testzwecken auch die von uns im ISIS zur Verfügung gestellte Beispiellösung nutzen. Bis auf Libraries des Spring Frameworks und Datenbank-Konnektoren dürfen keine weiteren Java Libraries genutzt werden.*

Spezifikation der Funktionalität

Channel erstellen: Channels mit Namen und Thema sollen durch Nutzer erstellt werden können. Namen für Channels sollen nicht doppelt vergeben werden können.

Channels abfragen: Die existierenden Channels sollen abgefragt werden können und als Liste zurückgegeben werden. Hier soll Pagination (s. letzter Punkt) umgesetzt werden, um zu große Anfragen zu verhindern.

Nutzer im Channel abfragen: Für jeden Channel soll eine Liste von Nutzern für diesen Channel abgefragt werden können. „Nutzer“ sind alle Verfasser von Nachrichten in diesem Channel (ohne Duplikate!).

Channel-Nachrichten abfragen: Der Server soll bei einer Abfrage der Nachrichten eines Channels die letzten 10 Nachrichten zurückliefern. Wenn bei der Abfrage ein Zeitstempel mitgeliefert wird, sollen stattdessen alle Nachrichten zurückgeliefert werden, die neuer oder gleich alt wie der mitgegebene Zeitstempel sind, maximal jedoch 50 Nachrichten (s. auch API-Beschreibung im Anhang).

Channel-Nachrichten schicken: Es soll möglich sein, Nachrichten für einen Channel an den Server zu schicken. Eine Nachricht besteht aus Nachrichtentext („content“), Autor („creator“) und einem Zeitstempel und ist einem Channel zugeordnet. Der Zeitstempel einer Nachricht soll vom Server festgelegt werden. Außerdem soll als Parameter beim Senden einer Nachricht ebenfalls ein Zeitstempel mitgegeben werden können. Als Antwort soll der Server dann alle seit dem Zeitstempel erstellten Nachrichten zurückgeben.

Persistenz: Der Server soll sämtliche Entitäten in einer Datenbank speichern. Nutzen Sie dazu eine lokale MySQL-Datenbank mit einer Konfiguration wie im Basis-Projekt gegeben (vgl. Datei *application.properties*). Beachten Sie, dass bei einem Neustart der Anwendung die Datenbank durch die Anweisung „create-drop“ stets geleert wird. Weitere Hinweise finden Sie auf den Folien zum Übungstermin 2.

Authentifizierung: Implementieren Sie eine serverseitige Authentifizierung, die für alle Requests prüft, ob ein gültiger Token als Value im http-Header mit dem Key „X-Group-Token“ vorhanden ist. Beim Fehlen des Tokens oder einem ungültigen Token soll ein entsprechender http-Statuscode zurückgeliefert werden. Tokens können Sie statisch beim Starten der Anwendung anlegen.

Pagination: Für die Abfragen von Channels und Nachrichten ist einfache Pagination wie folgt umzusetzen:

Beim Abrufen von Channel-Ressourcen sollen size und page vom Client dynamisch gesetzt werden können und die entsprechende Page als Antwort zurückgeliefert werden.

Beim Abrufen und Erzeugen von Nachrichten-Ressourcen sollen die Parameter *size* und *page* vom Server statisch gesetzt und in der Antwort entsprechend genutzt werden: Im Fall eines fehlenden „lastSeenTimestamp“ sollen 10 Nachrichten auf einer Page zurückgegeben werden. Wenn dieser gesetzt ist sollen stattdessen maximal 50 Nachrichten auf einer Page zurückgegeben werden. Blättern muss hier nicht möglich sein.

Nutzen Sie die Pagination Features von Spring-HATEOS.

Empfehlungen zur Vorgehensweise

- Überlegen Sie sich welche Entitäten Sie Modellieren müssen und implementieren Sie diese als POJOs und versehen Sie mit entsprechenden JPA-Annotationen.
- Schauen Sie sich die gegebene API-Dokumentation an und erstellen Sie entsprechende REST-Controller-Methoden für die unterschiedlichen Funktionen.
- Beginnen Sie damit, eine einzige Controller Methode (z.B. zum Abrufen aller Channels) zu erstellen und implementieren Sie diese vollständig. Testen Sie diese mit Postman und dem Frontend. Erweitern Sie erst dann die Funktionalität inkrementell um weitere Controller-Methoden.
- Nehmen Sie Fallunterscheidungen für unterschiedliche Parametrierung von Methoden vor. Fügen Sie passende Body-Objekte, Response Codes und Query Parameter ein, die Sie um die korrekten Annotationen ergänzen.

- Erstellen Sie nun notwendige Spring Data Repositories und implementieren Sie ggf. eigene notwendige Datenbankabfragen.
- Verbinden Sie die Repositories mit Ihren Controllern und testen Sie verschiedene Nutzungsszenarios.
- Kümmern Sie sich dann um Pagination und erst zuletzt um Authentifizierung.

Artefakte zur Abgabe

Dokumentieren Sie die Ergebnisse Ihrer Arbeit in einer Präsentation (Powerpoint o.ä.). Diese Präsentation muss für das Gesamtverständnis Ihrer Lösung genügen, insbesondere muss folgendes enthalten sein:

- Der Code muss ein valides Maven Projekt darstellen, d.h. das Anstoßen der Maven „package“ Lifecycle-Phase muss als Ergebnis ein lauffähiges *.jar-Archiv erzeugen.
- Eine Übersicht der von Ihnen erstellten Entities, Controller und Repositories, so wie eine Erklärung zu deren Zusammenhang.
- **Relevante** Codeauszüge mit hinreichender Beschreibung bzw. Dokumentation

Die Dokumentation stellt die wesentliche Bewertungsgrundlage dar, jedoch muss passender, lauffähiger Code vorhanden sein damit entsprechende Punkte erreicht werden können.

Wichtig: In der Abgabe darf die Datenbankkonfiguration gegenüber der Vorlage nicht verändert worden sein, damit wir Ihre Anwendung einfach testen können.

Das Deckblatt der Präsentation muss die Namen und Matrikelnummern der beteiligten Gruppenmitglieder beinhalten. Notieren Sie innerhalb der Artefakte, welche Gruppenmitglieder bei der Erstellung beteiligt waren.

Packen Sie ihren kompletten Code und alle zugehörigen Artefakte als .zip-Archiv. Reichen Sie die Präsentation (als .pdf!) und den Code (das Archiv) mit Hilfe des ISIS-Systems ein. Es genügt, wenn die Einreichung durch ein Gruppenmitglied erfolgt.

Zusätzliche Hinweise

- *Die Bearbeitung der zweiten Programmieraufgabe erfolgt in den gleichen Gruppen wie die der ersten Programmieraufgabe.*
- *Verwenden Sie als Grundlage für ihr Projekt die in ISIS zur Verfügung gestellte Vorlage, die auch in der Übung vorgestellt wurde.*
- *Nutzen Sie das Forum des ISIS-Kurses für technische Probleme, ggf. haben Ihre Kommilitonen ähnliche Probleme oder Sie finden dort bereits eine Lösung. Bei kritischen Problemen (die Sie auch nach gründlicher Recherche nicht beheben konnten) wenden Sie sich an einen Tutor.*
- *Quellcode, der übernommen wird, ist zu kennzeichnen und die Quelle anzugeben.*

Deadline für die Abgabe der Artefakte ist der 07.07.2019, 23.55 Uhr.

Beschreibung der HTTP-API des Servers

Folgend finden Sie eine Beschreibung der REST-API, die Ihr Server zur Verfügung stellen soll. Die Aufteilung ist nach Pfaden vorgenommen, bei Requests, bei denen Paramater übergeben werden können ist jeweils ein Beispiel aufgeführt. Sofern der Server bei der Response einen Response-Body mitschickt, ist dafür ebenfalls ein Beispiel gegeben.

/channels[?page,size]

Methoden:

a) GET

Beschreibung: Liefert eine Liste von Channels zurück. Bei Mehr als 20 existierenden Channels kann optional über den Query-Parameter (page=N) zur Seite N „geblättert“ werden. Alternativ kann die Seitengröße mit dem Parameter size=N erhöht werden. Die Anzahl der Seiten und die aktuelle Seitennummer kann dem page-Abschnitt des zurückgegebenen JSON-Bodys entnommen werden. Relative Pfadangaben zum Vorwärts-, bzw. Rückwärtsblättern sind außerdem im „_links“-Abschnitt vorhanden, wenn mehrere Seiten existieren.

Beispiel:

GET /channels

Responses: 200

Response Body (200):

```
{
    "_embedded": {
        "channelList": [
            {
                "id": 1,
                "name": "TestChannel",
                "topic": "All things testing"
            }
        ]
    },
    "_links": {
        "self": {
            "href": "/channels"
        }
    },
    "page": {
```

```
        "size": 20,
        "totalElements": 1,
        "totalPages": 1,
        "number": 0
    }
}
```

b) POST

Beschreibung: Erstellt eine neue Channel-Ressource, mit im Request-Body gegebenen Namen und Themenbeschreibung. Als Response erhält man einen Header mit dem relativen Pfad zur neu angelegten Ressource und im Body die angelegte Channel-Ressource. Falls ein Channel mit gleichem Namen existiert, wird ein Konflikt-Fehlerstatus zurückgegeben.

Beispiel:

POST /channels

Body: {

```
        „name“: „TestChannel“,
        „topic“: „All things testing“
    }
```

Responses: 201 (Header: Location), 409

Response Body (201): {

```
                    "id": 1,
                    "name": "testChannel",
                    "topic": "All things testing"
                }
```

/channels/{id}

Methoden:

a) *Beschreibung*: Liefert Informationen über den Channel mit gegebener ID zurück.

Beispiel:

GET /channels/1

Responses: 200, 404

Response Body (200): {

```
                    "id": 1,
                    "name": "testChannel",
                    "topic": "All things testing"
                }
```

/channels/{id}/messages[?lastSeenTimestamp]

Methoden:

a) GET

Beschreibung: Liefert Nachrichten aus dem über die ID gegebenen Channel zurück. Optional kann ein Parameter mitgegeben werden: lastSeenTimestamp. Bei weglassen der Parameter liefert der Server die jüngsten 10 Nachrichten des Channels. Bei Angabe von lastSeenTimestamp werden alle Nachrichten die neuer oder gleich alt als dieser übergebene Zeitstempel sind zurückgegeben. Achtung: Der Zeitstempel als Query-Parameter muss URL-encoded sein. Die Ergebnisse werden als Pages vom Server zurückgegeben, die Seitengrößen sind hier statisch vom Server festgelegt.

Beispiel:

GET /channels/1/messages?lastSeenTimestamp=2019-04-12T09%3A30%3A49.560247Z

Responses: 200, 404

Response Body (200):


```
{
    "_embedded": {
        "messageList": [
            {
                "id": 1,
                "timestamp": "2019-04-12T09:30:49.560247Z",
                "content": "Hallo Channel!",
                "creator": "user1",
                "channel": {
                    "@id": 1,
                    "id": 1,
                    "name": "TestChannel",
                    "topic": "All things testing"
                }
            }
        ]
    },
    "_links": {
        "self": {
```

```
            "href": "/channels/1"
        }
    },
    "page": {
        "size": 50,
        "totalElements": 1,
        "totalPages": 1,
        "number": 0
    }
}
```


b) POST

Beschreibung: Erstellt serverseitig eine neue Message-Ressource mit gegebenem Inhalt und Ersteller. Die erstellte Message ist dem Channel mit der ID aus dem Pfad zugeordnet. Auch hier kann der lastSeenTimestamp Query-Parameter genutzt werden. Er bewirkt, dass, zusätzlich zur neu erstellten Message gemäß des Requests, alle weiteren neu erstellten Messages von anderen Clients ebenfalls vom Server in der Response geliefert werden.

Beispiel:

POST /channels/1/messages?lastSeenTimestamp=2019-04-12T09%3A30%3A49.560247Z

Request-Body: {


```
          „creator“: „user1“,
          „content“: „Hallo Channel!“
    }
```


Responses: 200, 404

Response Body (202): {


```
    "_embedded": {
        "messageList": [
            {
                "id": 14,
                "timestamp": "2019-05-17T14:25:58.763698Z",
                "content": "Hallo Channel!",
                "creator": "user1",
                "channel": {
                    "@id": 1,
                    "id": 2,
```

```
                    "name": "testChannel",
                    "topic": "all things testing"
                }
            }
        ]
    },
    "_links": {
        "self": {
            "href": "/channels/1"
        }
    },
    "page": {
        "size": 100,
        "totalElements": 1,
        "totalPages": 1,
        "number": 0
    }
}
```


/channels/{id}/users

Methoden:

a) GET

Beschreibung: Liefert eine Liste aller Nutzern im Channel mit {id} zurück. Als Nutzer zählen alle Autoren der Nachrichten in diesem Channel.

Beispiel:

GET /channels/1/users


```
Responses: 200
Response Body (200): [
                    "user2",
                    "user1"
                ]
```