Anwendungssysteme (SS 2019)

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

Bewertete Programmieraufgabe (10 Portfoliopunkte)

Im Rahmen dieser Programmieraufgabe erstellen Sie mit Ihrer Gruppe einen Browser-basierten Client für eine Chat-Anwendung.

Aufgabenstellung

Ihre Aufgabe ist es das Client-seitige Interface (Frontend) der Chat-Anwendung *WhatsChat* zu entwickeln. Das WhatsChat-Frontend soll über eine graphische Oberfläche bedient werden und dem Nutzer die unten angegebene Funktionalität bereitstellen. WhatsChat ist vom UI-Design an den Instant-Messaging-Dienst Slack oder vergleichbaren Chatprogrammen angelegt: Es gibt Gruppenchats (*Channels*), in die Nutzer beitreten können, um dort Nachrichten auszutauschen. Das Frontend kommuniziert über eine http-basierte Programmierschnittstelle (API) mit dem Backend, das zentral vom Lehrstuhl bereitgestellt wird. Eine Beschreibung der API ist im Anhang des Dokuments zu finden.

Für die Umsetzung des Frontends ist HTML/CSS und JavaScript (jQuery) zu verwenden (vgl. Inhalte der Vorlesungen Web Technologies I+II). Es ist Ihnen freigestellt weitere JavaScript oder CSS-Frameworks zu nutzen. Bei anderen Frameworks als jQuery ist jedoch genau anzugeben welche Frameworks in welcher Version verwendet wurden. Außerdem ist dann eine hinreichende Dokumentation des Quellcodes (alle verwendeten Funktionen des Frameworks) erforderlich. *Eine Nutzung von Frameworks in anderen Sprachen, die nach JavaScript transpilieren (z.B. TypeScript), ist nicht zugelassen. Außerdem muss der Code ohne eine Installation von Server-Frameworks direkt im Browser lauffähig sein.*

Spezifikation der Funktionalität

Channels erstellen: Nutzer sollen Channels mit entsprechendem Namen erstellen können. Channels sind Chatgruppen. Jeder Channel hat ein Thema („topic“), das den Channel neben dem Namen charakterisiert. Serverseitig entspricht dies dem Anlegen einer neuen Channel-Ressource.

Channels darstellen: Die bestehenden Channels sollen in einer Liste im Frontend angezeigt werden. Über eine Liste kann ein Nutzer einen Channel auswählen, in dem er Nachrichten schreiben und lesen möchte. Die Liste der Channels soll in regelmäßigen Abständen aktualisiert werden (alle 10 Sekunden).

Channels bei- und austreten: Nutzer sollen einem Channel bei- und austreten können. Nur beigetretene Nutzer können Nachrichten des Channels lesen und Nachrichten an den Channel schreiben. Bevor ein Nutzer einem Channel beitritt, muss er einen Nutzernamen für den

jeweiligen Channel festlegen. Der Nutzer soll immer genau einem Channel beitreten können. Während der Nutzer einem Channel beigetreten ist sollen der Name und das Thema für diesen Channel angezeigt werden. Es soll außerdem in einer Liste angezeigt werden, welche anderen Nutzer sich zurzeit in diesem Channel befinden. Außerdem sollen die Channel-Nachrichten angezeigt werden (s. nächste Funktionalität).

Channel-Nachrichten anzeigen: Beim Betreten eines Channels sollen die jüngsten 10 Nachrichten dieses Channels im Nachrichtenfenster angezeigt werden. Die Logik dafür ist Server-seitig implementiert (siehe API Dokumentation).

Sobald der Nutzer den Channel betreten hat, soll der Nachrichtenverlauf regelmäßig (maximal sekündlich) mit dem Backend aktualisiert werden, d.h. der aktuell angezeigte Verlauf soll um die jüngsten Nachrichten in zeitlich korrekter Abfolge ergänzt werden. Dazu kann an die Anfrage der Zeitstempel der letzten im Verlauf sichtbaren Nachricht als Query-Parameter beigefügt werden (siehe auch API-Dokumentation).

Für jede Nachricht im Chatfenster sollen der Zeitpunkt der Veröffentlichung, der Autor und natürlich der Nachrichteninhalt angezeigt werden. Es darf keine Nachricht doppelt dargestellt sein.

Channel-Nachrichten schicken: Der Nutzer soll Nachrichten verfassen und diese in dem beigetretenen Channel veröffentlichen können. Eine Nachricht besteht aus Nachrichtentext („content“), Autor („creator“) und einem Zeitstempel und ist einem Channel zugeordnet. Autor und Channel sind dabei vorbestimmt durch den Channel, dem der Nutzer im Moment mit einem Nutzernamen beigetreten ist. Der Zeitstempel einer Nachricht wird Serverseitig festgelegt, sobald die Nachricht an den Server gesendet wurde. Beachten Sie auch hierzu die Beispiele in der API Dokumentation.

Erfolgreich an den Server gesendete Nachrichten sollen direkt nach dem Senden an die bereits vorliegenden Nachrichten im Nachrichtenfenster angehängt werden.

Empfehlungen zur Vorgehensweise

- Überlegen Sie sich welche HTML-Elemente bzw. Seiten Sie benötigen, um die geforderten Funktionalitäten darzustellen.
- Schauen Sie sich die gegebene API-Dokumentation an und überprüfen Sie welche Ressourcen, Methoden und Pfade sie wo auf ihrer Webseite einbinden wollen
- Führen Sie einfache Beispielaufrufe an die API durch (z.B. zunächst mit Postman, dann mit JavaScript aus dem Browser) um ein Gefühl dafür zu bekommen, wie die Daten, die Sie vom Server bekommen, aussehen. Achten Sie darauf, dass Sie die Pfade und Felder gemäß der API-Dokumentation nutzen
- Implementieren Sie die nötigen HTTP-Aufrufe in JavaScript
- Verbinden Sie ihr HTML-Gerüst mit den JavaScript Methoden in geeigneter Weise
- Kümmern Sie sich erst zum Ende um eine passende, bzw. schöne Darstellung durch den Einsatz von CSS

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:

- Eine Übersicht der von ihnen erstellten Seiten, Elemente und eine Erklärung der Nutzung bzw. Navigation
- **Relevante** Codeauszüge mit hinreichender Beschreibung bzw. Dokumentation
- (Gut erkennbare!) Screenshots der Web-Darstellung aller Seiten Ihrer finalen Anwendung mit Zuordnung zur jeweiligen Funktionalität

Die Dokumentation stellt die wesentliche Bewertungsgrundlage dar, jedoch muss passender, lauffähiger Code vorhanden sein damit entsprechende Punkte erreicht werden 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 Programmieraufgabe erfolgt in den über ISIS gebildeten Gruppen. Prüfen Sie ggf. über ISIS wer Mitglied ihrer Gruppe ist.*
- *Verwenden Sie als Grundlage für ihr Projekt die in ISIS zur Verfügung gestellte Vorlage, die auch in der Übung vorgestellt wurde.*
- *Um die Programmierschnittstelle des Servers nutzen zu können müssen Sie sich beim Server authentifizieren. Dazu erhalten Sie als Gruppe eine E-Mail mit ihrem sog. Group-Token. Dieses Token müssen Sie als Wert des http-Header-Felds („X-Group-Token“) bei jedem Request mitschicken.*
- *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 am 16.06.2019, 23.55 Uhr.

Beschreibung der HTTP-API des Servers

Folgend finden Sie eine Beschreibung der Serverseitig zur Verfügung gestellten http-Methoden. 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. Beachten Sie, dass für alle Requests das Header-Feld „X-Group-Token“ gesetzt sein muss. **Der Server erwartet JSON-Dokumente als Request-Body und schickt JSON-Dokumente als Response-Body zurück.**

Adresse des Servers: **http://34.243.3.31:8080**

Authentication-Header (Beispiel): „X-Group-Token: Token1234567"

Hinweis: *Die Datenbank des Servers wird in regelmäßigen Abständen (i.d.R. am frühen Vormittag) zurückgesetzt, um einer übermäßigen Last bzw. zu großen Anfragen und Antworten bei der Kommunikation vorzubeugen.*

/channels[?page,size]

Methoden:

a) *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) *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, bzw., falls ein Channel mit gleichem Namen existiert, einen Konflikt-Fehlerstatus zurück.

Beispiel:

POST /channels

Body: {

„name“: „TestChannel“,

„topic“: „All things testing“

}

Responses: 201 (Header: Location), 409

/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,page]

Methoden:

a) *Beschreibung*: Liefert Nachrichten aus dem über die ID gegebenen Channel zurück. Optional können zwei Parameter mitgegeben werden: lastSeenTimestamp und page. 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. Der Page-Parameter ermöglicht das Blättern, sofern mehr als 50 Nachrichten zurückgegeben werden. Die Seitengrößen sind hier statisch vom Server festgelegt.

   Beispiel:

   GET /channels/1/messages?lastSeenTimestamp=2019-04-12T09%3A30%3A49.560247Z&page=0

   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) *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) *Beschreibung*: Liefert eine Liste aller im Channel mit {id} aktiven Nutzernamen zurück. Ein aktiver Nutzer ist einer, der innerhalb der letzten 10 Minuten eine Nachricht an den Channel gesendet hat.

Beispiel:

GET /channels/1/users

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