BAI5-VS	Praktikum Verteilte Sicherheit – Aufgabenblatt 1	MDE/KSS/ZGR
WiSe 2024/25	Stern-Topologie – Infrastruktur und Messaging	

Korrekturen für Version 1a (5. November 2024):

- In verschiedenen REST-Aufrufen wird der Parameter „`sol=<STAR-UUID>`“ Parameter verwendet. Es muss heißen: „`star=<STAR-UUID>`“ und dies wurde in allen Aufrufen entsprechend geändert.
- Im vorletzten Absatz von 1.1 (`GET /vs/v1/system/<COM-UUID>?star=<STAR-UUID>`) wird ein Fehlerzustand beschrieben, der nicht auftreten kann, wenn die vorherigen Prüfungen korrekt abgearbeitet wurden. Ein Satz ist gestrichen.

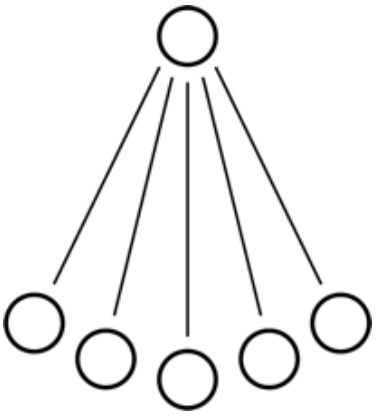
Sie programmieren eine Peer-To-Peer-Anwendung, d.h. alle Komponenten sind erstmal potenziell gleich, manchmal wird eine oder werden mehrere mit der Zeit besonderer als andere, aber eins nach dem anderen ;) Ihre Peers sollen, wenn sie aktiviert werden, sich als Teil eines sogenannten Systems-of-Systems von gleichartigen Peers integrieren, hier also Teil eines verteilten System in einem lokalen (Computer-) Netzwerk werden.

Dafür müssen Sie verschiedene Herausforderungen lösen:

- Gibt es beim Komponentenstart bereits ein System, in das wir uns integrieren können?
- Wenn wir eins finden: wie kommen wir in das System rein?
- Wenn wir evtl. mehrere finden: in welches integrieren wir uns?
- Wenn wir ein neues System starten: wie können sich andere integrieren?
- Kommen wir auch wieder raus, wenn wir reingekommen sind? Und dürfen wir das überhaupt?
- Wie kommen die anderen damit zurecht, dass eine Komponente rausgeht?
- Und bekommt unsere Komponente überhaupt mit, ob die anderen noch da sind?
- Und wer macht am Ende das Licht aus?

Später muss gelöst werden, was unsere Komponente denn überhaupt macht, wenn das verteilte System am Laufen und unsere Komponente integriert ist.

Welche Anwendung soll nun angeboten werden?



Wir wollen ein Messaging-System entwickeln, das in einer Stern-Topologie funktioniert! (Mehr zu Topologien im Foliensatz 03). Warum nun die Stern-Topologie? Weil es sehr nahe am (bekannten) Client-Server-Ansatz ist und die Topologie gut zur Anwendung passt. (Oder genauer gesagt: zu einer der beiden vorrangigen Realisierungsoptionen passt!)

Ach ja, immer dran denken, das Internet hat seine eigene technische Topologie (teilvermascht, wie wir das in Rechnernetzen kennengelernt haben) und wir benutzen diese, um „unsere“ anwendungsspezifische Topologie „darüber“ aufzubauen, zu verwalten und vor allem um Nachrichten zwischen den Peer-Komponenten auszutauschen. Hierbei handelt es sich um eine logische Topologie oder „Overlay“ , welche die bestehenden Netzwerke nutzen.

Was sind die Rahmenbedingungen für Ihre Programmierung?

Auf ein paar Sachen muss man sich einigen. Anfangen werden wir mit dem Standard <STARTPORT> zur Discovery der Services arbeiten, die auf der <ID> Ihrer Arbeitsgruppe beruht:

<STARPORT> ::= 8000 + <ID> = 8013

Dieser Port soll jedoch variabel bleiben, so dass leicht auf andere Ports umgeschwenkt werden kann, um nicht von unerwarteten Nachrichten gestört zu werden. Also nicht fest einkodieren!

Für die Kommunikation mit der Komponente sind fünf Schnittstellen vorzusehen:

1. Beim Aufruf durch den Menschen vor dem Bildschirm gibt es einige wenige Parameter, die Sie aber auch eigenständig erweitern können.

2. Die Interaktion mit dem Menschen vor dem Bildschirm während der Laufzeit ist sehr begrenzt und kennt sehr weniger Befehle:
   a. „CRASH“ : wird dieser Befehl eingegeben, beendet sich das Programm **ALS OB ES DURCH EINEN CRASH MITTEN IN DER AUSFÜHRUNG UNTERBROCHEN WORDEN WÄRE!** Es werden weder Aufgaben zu Ende bearbeitet noch Verbindungen abgebaut. Wir können also typische Ausfälle und Fehlerfälle im Netzwerk „einspielen“.
   b. „EXIT“ : wird dieser Befehl eingegeben, meldet sich das Programm beim Stern ab (siehe unten, Einzelheiten abhängig von der Rolle der Komponente) und beendet sich dann selbst.

3. Die Protokollierung von wichtigen Ereignissen über den Programmablauf und vor allem die Interaktion mit anderen Komponenten soll via SYSLOG erfolgen, so dass diese durch den Menschen vor dem Bildschirm mit entsprechendem lesenden Zugriff auf die (SYS)Log-Datei geprüft und beobachtet werden können.

4. Die BROADCAST-Kommunikation per UDP „mit allen aktiven Komponenten im Netzwerk“ (auf das lokale Netzwerk beschränkt), wobei Pakete an den <STARPORT>/udp gesendet werden müssen.

   Sofern nichts anderes beschrieben wird, enthalten die Pakete für die BROADCAST-Kommunikation nur eine Zeichenkette von max. 1024 Zeichen in UTF-8. Diese Zeichenkette muss durch Null-Byte terminiert sein.[1]

5. Die UNICAST-Kommunikation per TCP „mit Einzelnen aber potenziell allen aktiven Komponenten im Netzwerk“, wobei Verbindungen zu einem gegebenen Port <STARPORT>/tcp aufgebaut werden.

---

[1] Achtung! Wenn Nachrichten an die Broadcast-Adresse 255.255.255.255 gesendet werden, so wird diese NICHT auf der IP empfangen, an die der sendende Socket selbst gebunden ist. Wenn also ein eigener BROADCAST nicht ankommt, so hilft evtl. an eine „kleinere“ Broadcast-Domain zu senden wie z.B. 127.255.255.255. (https://stackoverflow.com/a/74943642) Da Sie ja aber die richtige NETMASK nutzen, kommt das Problem in der Praxis hoffentlich nicht vor …

Zur Identifikation von Komponenten werden UUIDs benutzt, deren Nutzung dann mit der Zeit deutlich(er) wird.

<STAR-UUID> : für die Identifizierung eines bestimmten Sterns
<COM-UUID> : für die Identifizierung einer bestimmten Komponente, eindeutig innerhalb eines bestimmten Sterns

1.0 Erst mal reinkommen …

Um reinzukommen in den Stern, muss die Komponente zunächst etwas rausbekommen, nämlich ob bereits ein Stern aktiv ist! Eine neu gestartete Komponente „kennt“ zunächst niemanden. In den Wald reinbrüllen geht immer, und wenn schon ein Stern aktiv ist, ruft jemand hoffentlich auch zurück!

Übersetzt heißt das: per BROADCAST anfragen, ob ein Stern aktiv ist! Hierzu wird die Zeichenkette „HELLO?“ in einem UDP-Paket gesendet.

```
Aufruf:         “HELLO?”
```

Aktive Sterne müssen auf Broadcasts an <STARPORT>/upd lauschen und im Falle einer „HELLO?“-Nachrichten mit folgenden Informationen (also einem JSON-Blob) antworten. Diese Antwort wird jedoch direkt an den <STARPORT>/upd der neuen Komponente gesendet mit relevanten Informationen:

```
Antwort von SOL:
                {
                     “star”          : <STAR-UUID>,
                     “sol”           : <COM-UUID>,
                     “sol-ip”        : <IP>,
                     “sol-tcp”       : <STARPORT>,
                     “component”     : <COM-UUID>
                }
```

Hierbei wird in der Antwort die <STAR-UUID> des Sterns übermittelt, sowie die Angaben der Komponente, die den Stern betreibt (auch eine <COM-UUID>, deren IP und Portnummer für TCP). Ach ja, diese antwortende Komponente nennen wir „SOL“. Die Komponenten könnten wir als Planeten bezeichnen, aber das lassen wir lieber …

Es könnte sein, dass mehrere Sterne aktiv sind und somit mehrere Antworten eintreffen. Die neue Komponente kann sich dann aussuchen, bei wem sie Teil werden möchte, aber (bis auf weiteres) immer nur bei einem. (Vorsicht Falle! Hatten wir in der Vorlesung 😉)

Die neue Komponente verwendet die übermittelten IP-Adresse sowie Port-Angabe und kann sich mit diesen Angaben bei der zentralen Komponente des Sterns über einen Aufruf der REST-API per UNICAST registrieren (übergebene Daten sind übrigens immer vom Content-Type „application/json“ sofern nicht anders angegeben).

Bei der Integration sind <COM-UUID>, <IP> und <PORT> die Werte der neuen Komponente und SOL muss diese – aber auch erst dann – entweder in seine Liste aufnehmen, oder die Registrierung ablehnen. Damit SOL den auch kennt, gibt die neue Komponente selbst ihren derzeitigen Status an. Außerdem werden Angaben zum Stern übermittelt, die von SOL vor der Integration auch geprüft werden, damit sichergestellt ist, dass auch der „richtige“ Stern gemeint ist.

```
Aufruf:         POST /vs/v1/system/

Übergebene Daten:
                {
                     “star”          : <STAR-UUID>,
                     “sol”           : <COM-UUID>,
                     “component”     : <COM-UUID>,
                     “com-ip”        : <IP>,
                     “com-tcp”       : <PORT>,
                     “status”        : <STATUS>
                }

Ergebnis:       text/plain

Antworten:      200 ok
                401 unauthorized
                403 no room left
                409 conflict
```

Der <STATUS> verwendet Response Codes wie wir sie auch für die Antworten selbst verwenden und z.B. von HTTP kennen, siehe Appendix A für eine Liste verwendeter Codes. Bei der Integration wird zunächst immer „200“ gemeldet, um zu signalisieren, dass sie weiterhin bereit ist. Was anderes wäre ja seltsam …

Wenn die Angaben zum Stern nicht stimmen, wird mit „401“ die Registrierung abgelehnt. Mit „403“ wird die Registrierung abgelehnt, wenn der Stern „voll“ ist (siehe unten). Wenn die Angaben zur Komponente, die integriert werden möchte, nicht stimmen, wird mit „409“ die Registrierung ebenfalls abgelehnt. Bei einer Ablehnung muss sich die nicht integrierte Komponente selbst ohne weitere Aktionen beenden.

Im anderen Fall, also wenn kein Stern aktiv ist bzw. sich niemand sofort meldet, wird nach weiteren 10 bzw. 20 Sekunden der „Brüller“ wiederholt. Erst, wenn sich dann immer noch kein Stern gemeldet hat, weiß die Komponente:

Huch, ich bin die erste?

Eigentlich ist es ziemlich einfach, wenn es noch keinen Stern gibt. Die Komponente bildet dann selbst den „Mittelpunkt“ des neuen Sterns („SOL“) und nimmt ab diesem Zeitpunkt die neuen „Brüller“ per BROADCAST an, beantwortet diese und integriert damit neue Komponenten.

Aber erstmal ganz langsam, der Mittelpunkt des neuen Sterns muss selbst ein paar Sachen initialisieren, bevor es weitergehen kann:

- die eigene <COM-UUID> und ja, die muss selbst berechnet werden
- Zeitstempel der Initiierung des Sterns
- die <STAR-UUID>, die selbst berechnet werden muss
- Anzahl der aktiven Komponenten ::= „1“
- Maximale Anzahl der aktiven Komponenten ::= „4“ (auch das muss ein Aufrufparameter werden)

SOL muss sich selbst als Teil des Sterns abspeichern, hier sind die folgenden Angaben relevant – gilt natürlich auch für alle weiteren Komponenten des Sterns:

- <COM-UUID>
- IP-Adresse
- Der verwendete TCP-Port
- Zeitstempel der Integration in den Stern
- Zeitstempel der letzten Interaktion mit dem Stern

Wie werden die Angaben für die UUIDs nun generiert?

<COM-UUID> ::= int(rand(9000) + 1000)
<STAR-UUID> ::= md5(IP-Adresse von SOL, <ID>, <COM-UUID> von SOL)

Also ist die <COM-UUID> eine vierstellige Zahl größer 999 und wird zufällig vergeben. Es kann allerdings passieren, dass eine später erzeugte <COM-UUID> schon einer aktiven Komponente zugeordnet wurde, d.h. bevor ein erzeugter Wert verwendet werden kann, muss dies geprüft werden. Ggf. ist eine neue Zufallszahl zu ermitteln.

1.1 Pflege des Sterns – Kontrolle der Komponenten

Jede aktive Komponente baut alle 30 Sekunden eine UNICAST-Verbindung zum <STARPORT>/tcp von SOL auf.[2] Wenn SOL nicht erreichbar ist, wird es nach 10 bzw. 20 Sekunden nochmal versucht. Wenn dann immer noch keine Verbindung zustande kommt, beendet sich die Komponente selbst.[3]

```
Aufruf:         PATCH /vs/v1/system/<COM-UUID>

Übergebene Daten:
                {
                     “star”          : <STAR-UUID>,
                     “sol”           : <COM-UUID>,
                     “component”     : <COM-UUID>,
                     “com-ip”        : <IP>,
                     “com-tcp”       : <PORT>,
```

[2] Hier können Sie auch entscheiden, dass SOL selbst an diesem Verfahren teilnimmt. SOL mit aufzunehmen bedeutet, dass die Schleifen einfacher werden, weil keine Ausnahmen zu beachten sind. Bewerten Sie selbst, wie Sie es machen wollen.

[3] Diese Timeouts und das danach erfolgende „Runterfahren“ kann das Testen erschweren, überlegen Sie sich was, wie Sie bestimmte Funktionalitäten zum Testen „abschalten“ können oder schnell – ohne Programmierung oder „build“ ändern können. Nur vergessen Sie nicht, dass die abgeschalteten Funktionalitäten später auch schnell wieder angeschaltet werden sollen. 😉

```
                    "status"       : <STATUS>
               }

Ergebnis:      text/plain

Antworten:     200 ok
               401 unauthorized
               404 does not exist
               409 conflict
```

Die Angaben sind aus dem POST-Aufruf weiter oben bereits bekannt. Geht diese Anfrage an eine andere Komponente als SOL, wird diese mit „401“ ablehnen.

SOL prüft, ob die IP-Adresse der Komponente noch der gespeicherte IP entspricht, um welchen Stern es sich handelt und die übermittelten <COM-UUID> sowie IP und Port-Angaben stimmen. Wenn ja, wird der aktuelle Status und der Zeitpunkt dieser Meldung abgespeichert und „200“ als Antwort geliefert. Sonst wird „401“ geliefert, wenn Angaben für den Stern bzw. SOL nicht stimmen, oder „409“, wenn die Angaben zur Komponente nicht stimmen oder ein <STATUS> ungleich „200“ übermittelt wurde. „404“ wird geliefert, wenn die Komponente nicht im Stern registriert ist.

Wenn SOL für eine aktive Komponente 60 Sekunden nach der letzten Meldung keine neue Meldung mit einem Status von „200“ erhält, baut SOL zum <STARPORT>/tcp der Komponente eine UNICAST-Verbindung auf und kontrolliert selbst, ob die Komponente noch aktiv und funktionsfähig ist. Diese Kontrollmöglichkeit muss SOL auch für sich selbst unterstützen! Auch hier kommt eine REST-API zum Einsatz.

```
Aufruf:        GET /vs/v1/system/<COM-UUID>?star=<STAR-UUID>

Ergebnis:      application/json

Antworten:     200 ok

               {
                    "star"         : <STAR-UUID>,
                    "sol"          : <COM-UUID>,
                    "component"    : <COM-UUID>,
                    "com-ip"       : <IP>,
                    "com-tcp"      : <PORT>,
                    "status"       : <STATUS>
               }

               401 unauthorized
               409 conflict
```

Wenn die Komponente noch aktiv ist, prüft diese:

1. ob sie selbst im angegebenen Stern ist, sonst „401“. Dies gilt auch bei leerer <COM-UUID> beim GET-Aufruf.

2. ob – sofern die angegebene <COM-UUID> nicht leer ist – die Komponente korrekt referenziert wird, wenn nicht „409“.

Wenn beide Prüfungen „klappen“, wird mit „200“ der aktuelle Status übermittelt. SOL speichert den aktuellen Status und den Zeitpunkt der Antwort. D.h. eine Komponente kann nur den eigenen Status melden. ~~Wird eine Komponente nach einer fremden <COM-UUID> befragt, wird „401“ geantwortet.~~

Wenn die Komponente nicht erreicht werden kann, wird die Komponente aus dem Stern entfernt. Der Dateneintrag bleibt aber erhalten mit der Angabe „disconnected“, falls später die Komponente sich doch noch mal melden sollte.

1.2 Pflege des Sterns – Abmelden einer Komponente

Eine aktive Komponente, die sich nach einem „EXIT“-Befehl bei SOL abmeldet, baut eine UNICAST-Verbindung auf. Wenn SOL nicht erreichbar ist, wird es nach 10 bzw. 20 Sekunden nochmal versucht. Wenn dann immer noch keine Verbindung zustande kommt, beendet sich die Komponente selbst.

```
Aufruf:         DELETE /vs/v1/system/<COM-UUID>?star=<STAR-UUID>

Ergebnis:       text/plain

Antworten:      200 ok
                401 unauthorized
                404 not found
```

Jede Komponente darf sich selbst bei SOL abmelden (inklusive SOL, z.B. beim Shutdown). Andere Komponenten ignorieren diese Abmeldung mit „401“.

SOL prüft, ob die Anfrage zur Löschung von der hinterlegten IP stammt und es sich um den richtigen Stern handelt. Ist dies nicht der Fall, wird die Abmeldung mit „401“ verweigert. Soll eine Komponente abgemeldet werden, die nicht bei SOL als aktiv gespeichert ist, wird „404“ geantwortet.

Stimmt alles, wird der aktuelle Status und der Zeitpunkt dieser Meldung durch SOL abgespeichert. Die Komponente wird aus dem Stern entfernt. Der Dateneintrag bleibt aber erhalten mit der Angabe „left“.

Die sich abmeldende Komponente beendet sich selbst, auch bei einem Statuscode, der einen Fehler signalisiert.

1.3 Pflege des Sterns – Abmelden von SOL

Wenn die Komponente, die gerade aktiv den Stern „managed“ (also SOL) den „EXIT“-Befehl bekommt, werden von ihr alle aktiven Komponenten im Stern einzeln kontaktiert:

```
Aufruf:        DELETE /vs/v1/system/<COM-UUID>?star=<STAR-UUID>

Ergebnis:      text/plain

Antworten:     200 ok
               401 unauthorized
```

Die angesprochene Komponente prüft die Angaben für SOL und ignoriert den Aufruf, wenn die Angaben nicht stimmen mit „401“. Sonst antwortet die Komponente mit „200“ und stellt ihre Funktion ein, als ob „EXIT“ eingegeben wurde.

Wenn eine Komponente aus dem Stern entfernt wird, wird deren Dateneintrag auf „disconnected“ gesetzt.

Wenn eine Komponente nicht erreichbar ist, wird es nach 10 bzw. 20 Sekunden nochmal versucht. Wenn dann immer noch keine Verbindung zustande kommt, wird es nicht weiter versucht.

SOL beendet sich selbst erst, wenn alle aktiven Komponenten erreicht wurden bzw. für nicht antwortende Komponenten beide Wiederholungen fehlschlugen.

2.1 Nachrichten können erzeugt und gespeichert werden

Alle Komponenten, die in den Stern integriert sind – also auch SOL, können Nachrichten erzeugen – oder dazu gebracht werden, Nachrichten zu erzeugen. Nachrichten müssen durch die Komponenten immer an SOL weitergegeben werden. SOL sorgt dann dafür, dass diese Nachrichten für alle aktive Komponenten zur Verfügung stehen, d.h. dort abgerufen werden können. Nachrichten werden anhand eindeutiger Nachrichten-UUIDs identifiziert. Geänderte Nachrichten werden anhand von Versionsnummern weiter differenziert.

Anstelle jedoch eine GUI für die Eingabe von Nachrichten zu fordern, können wir die gleichen Schnittstellen auch benutzen, um Nachrichten z.B. über die SWAGGER UI (Appendix C) zu erzeugen, zu verändern und zu lesen – wir nutzen also die API einfach auch als Userinterface. Im Mittelpunkt der Funktionalität steht folgender Aufruf, der sowohl für die Erstellung einer neuen Nachricht als auch für das Weiterleiten an SOL genutzt werden muss:

```
Aufruf:        POST /vs/v1/messages/<MSG-UUID>

Übergebene Daten:
               {
                    “star”    : <STAR-UUID> ,
                    “origin”  : <COM-UUID> | <EMAIL> ,
                    “sender”  : <COM-UUID> | “” ,
                    “msg-id”  : <MSG-UUID> | “” ,
                    “version” : “1” | “” ,
                    “created” : <TIMESTAMP> ,
                    “changed” : <TIMESTAMP> ,
```

```
                    "subject" : <STRING> ,
                    "message" : <STRING>
                }

Ergebnis:       text/plain

Antworten:      200 ok
                401 unauthorized
                404 already exist
                412 precondition failed
```

Anhand der <STAR-UUID> wird sichergestellt, dass Nachrichten auch für diesen Stern „gedacht" sind. Ist die Komponente nicht im „richtigen" Stern, gibt es die Antwort „401" und die Nachricht wird nicht akzeptiert.

<EMAIL> ist eine Standard-Email-Adresse und gibt an, welcher Mensch oder Prozess diese Nachricht erzeugt hat bzw. dafür verantwortlich ist. Alternativ kann auch eine Komponente als Urheber auftreten, angezeigt durch die <COM-UUID>.

Das Subject der Nachricht und der „Body" der Nachricht sind Zeichenketten in UTF-8. Diese können alle Zeichen enthalten, die einzige Einschränkung gibt es für das Subject. Dieses wird zwar in beliebiger Länge angenommen, aber bei der Weiterverarbeitung (Weiterleiten, Speicherung, ...) gekürzt, und zwar bis zum ersten NEWLINE-Zeichen.[4] Alle „CARRIAGE RETURN"-Zeichen werden vor der weiteren Verarbeitung aus dem Betreff gelöscht.[5]

Sind „Origin" oder das Subject der Nachricht leer oder stimmt das Format der Daten nicht, wird mit „422" geantwortet und die Nachricht wird nicht akzeptiert. Ein leerer Body ist akzeptabel. Angenommene Nachrichten werden mit „200" angezeigt.

Die <MSG-UUID> wird leer gelassen. Nachrichten mit leerer <MSG-UUID> werden mit dem gleichen API-Aufruf wie oben an SOL gegeben. SOL bearbeitet die Nachricht und setzt dann bei der Speicherung die <MSG-UUID> auf einen noch nicht vergebenen Wert. Die Version einer Nachricht beginnt immer bei „1".

<MSG-UUID> ::= <NUMBER> . „@" . <COM-UUID>

<NUMBER> ist dabei ein einfacher Zähler, der beim Start des Sterns mit „1" initialisiert wird und bei neuen Nachrichten hochgezählt wird. <COM-UUID> ist entweder die unter <ORIGIN> eingetragene <COM-UUID>-Angabe oder die <COM-UUID> des Senders.

Allerdings erlaubt der Aufruf auch, eine (nicht leere) <MSG-UUID> zu verwenden. Diese wird auch an SOL übermittelt und wird akzeptiert, wenn die <NUMBER> noch nicht vergeben ist. In einem solchen Fall wird auch der Zähler auf diesen Wert gesetzt, d.h. die nächste regulär vergebene <NUMBER> wäre um eins höher als der übergebene Wert. Auch hier ist die Antwort „200".

---

[4] Wird manchmal auch als „Line Feed" bezeichnet, ist 0x0A oder „\n".

[5] Ist 0x0D oder ‚\r'.

Gibt es die <MSG-UUID> schon, wird „404“ geantwortet und die neue Nachricht ignoriert.

Wird eine Nachricht von SOL akzeptiert, werden die Angaben zu „status“, „created“ und „changed“ gesetzt. Hierbei werden ggf. übergebene Werte nicht verwendet:

„status“ ::= „active“
„created“ ::= aktueller Zeitstempel in UNIX-Notation[6]
„changed“ ::= gleicher Zeitstempel wie „created“

2.2 Nachrichten können gelöscht werden

Alle Komponenten, die in den Stern integriert sind, können Löschaufträge annehmen. Löschaufträge müssen immer an SOL weitergegeben werden. SOL sorgt dann dafür, dass der Inhalt gelöschter Nachrichten nicht mehr zur Verfügung steht, aber die Meta-Daten erhalten bleiben. Löschungen funktionieren mit folgendem Aufruf, der wieder bei SOL und anderen Komponenten identisch ist:

```
Aufruf:         DELETE /vs/v1/messages/<MSG-UUID>?star=<STAR-UUID>

Ergebnis:       text/plain

Antworten:      200 ok
                401 unauthorized
                404 does not exist
```

Anhand der <STAR-UUID> wird sichergestellt, dass nur Nachrichten aus diesem Stern „gelöscht“ werden. Ist die Anfrage nicht für den „richtigen“ Stern, gibt es die Antwort „401“ und der Auftrag wird nicht akzeptiert. Ebenso nicht, wenn die <MSG-UUID> leer ist oder nicht gefunden wird – quittiert mit 404.

Wenn SOL einen Löschauftrag annimmt und die Nachricht löscht, wird „200“ zurückgegeben. SOL speichert den Zeitpunkt der Löschung und ändert den Status:

„status“ ::= „deleted“
„changed“ ::= Zeitstempel des Auftrags

2.3 Die Liste aller Nachrichten kann von SOL geholt werden

Alle Komponenten, die in den Stern integriert sind – also auch SOL, können die Liste der Nachrichten abfragen. Aber es gibt unterschiedliche Möglichkeiten, wenn gewünscht können weitere Informationen der gültigen Nachrichten gleich mit abgerufen werden. Und es können auch die gelöschten <MSG-UUID>s mit abgerufen werden – hierfür gibt es allerdings keine weiteren Informationen.

```
Aufruf:         GET /vs/v1/messages?star=<STAR-UUID>& \
                                    scope=<SCOPE>&info=<INFO>
```

[6] 4 Byte-Wert, der die Zahl der Sekunden seit dem 1. Januar 1970 (UTC) angibt.
siehe auch: https://www.unixtimestamp.com (und man beachte den Hinweis am Ende der Seite!)

```
     <SCOPE> ::= “active” (default) | “all”
     <INFO>  ::= “id” (default) | “header”

Ergebnis:      application/json

Antworten:     401 unauthorized
               200 ok

// leere Liste
     {
          “star”:          <STAR-UUID>,
          "totalResults": 0,
          "scope":         <SCOPE>,
          "view":          <VIEW>,
          “messages”: []
     }

// nicht leere Liste, auch gelöschte, nur <MSG-UUID>s
     {
          “star”:          <STAR-UUID>,
          "totalResults": 2,
          "scope":         „all“,
          "view":          „id“,
          “messages”: [
               {
                    “msg-id” : “2@9573”
                    “status” : “active”
               },
               {
                    “msg-id” : “6@3711”
                    “status” : “deleted”
               }
          ]
     }

// nicht leere Liste, auch gelöschte und alle Header
     {
          “star”:          <STAR-UUID>,
          "totalResults": 2,
          "scope":         „all“,
          "view":          „header“,
          “messages”: [
               {
                    “msg-id”:  “2@9573”
                    “version”: “1”,
                    “status”:  “active”
                    “origin”:  “9573”,
                    “created”: 1726788854,
                    “changed”: 1726788854,
                    “subject”: “VS Praktikum ist voll”
               },
```

```
                {
                     “msg-id”:  “6@3711”
                     “status”:  “deleted”
                }
          ]
     }
```

Anhand der <STAR-UUID> wird sichergestellt, dass Nachrichten von diesem Stern „gemeint“ sind. Ist die Komponente nicht im „richtigen“ Stern, gibt es die Antwort „401“ und der Auftrag wird nicht akzeptiert.

2.3 Eine Nachrichten kann von SOL geholt werden

Alle Komponenten, die in den Stern integriert sind – also auch SOL, können bei Eingabe einer gültigen <MSG-UUID> diese von SOL erhalten.

```
Aufruf:        GET /vs/v1/messages/<MSG-UUID>?star=<STAR-UUID>

Ergebnis:      application/json

Antworten:     401 unauthorized
               404 does not exist

// <MSG-UUID> ist leer bzw. es wurde keine übergeben
// Nachricht liegt nicht vor
// oder Anfrage ist unautorisiert (<STAR-UUID> ist falsch)
     {
          “star”:         <STAR-UUID>,
          "totalResults": 0,
          “messages”: []
     }

               200 ok

// gelöschte <MSG-UUID>
     {
          “star”:         <STAR-UUID>,
          "totalResults": 1,
          “messages”: [
               {
                    “msg-id” : “6@3711”
                    “status” : “deleted”
               }  ]
     }
// nicht gelöschte <MSG-UUID>
     {
          “star”:         <STAR-UUID>,
          "totalResults": 1,
          “messages”: [
```

```
                {
                     "msg-id":  "2@9573"
                     "version": "1",
                     "status":  "active"
                     "origin":  "9573",
                     "created": 1726788854,
                     "changed": 1726788854,
                     "subject": "VS Praktikum ist voll",
                     "message": "Aber es gibt noch Chancen"
                }  ]
     }
```

Anhand der <STAR-UUID> wird sichergestellt, dass Nachrichten von diesem Stern „gemeint“ sind. Ist die Komponente nicht im „richtigen“ Stern, gibt es die Antwort „401“ und der Auftrag wird nicht akzeptiert. Auch wenn die <MSG-UUID> nicht existiert, wird eine leere Liste zurückgegeben und die Antwort „404“. Ebenso wird „404“ zurückgegeben, wenn die angesprochene Komponente eben nicht SOL ist.

Appendix A: Response Codes

Nicht abschließend gemeint, aber erstmal ein guter Anfang …

Response Code	Bedeutung (mehrere Interpretationen möglich)
200	success | ok
201	created
400	invalid input
401	unauthorized
403	no room left | forbidden
404	already exist | does not exist | not found
409	conflict
412	precondition failed
422	unprocessable content

Appendix B: Aufrufe mit CURL

Mit dem Programm CURL kann man sehr einfach REST-Schnittstellen testen, ohne einen Client „haben“ zu müssen. Dies geht z.B. so, dabei unbedingt an den richtigen TCP-Port denken (hier für die Gruppe mit der <ID> == „13“):

```
% curl -H "Life Signal" \
       -X PATCH "http://123.123.123.17:8013//vs/v1/system/4567
       -d '{  "star": 0AF1...EF, \
              "sol": "1234", \
              "component": 4567, \
              "com-ip": 123.123.123.125, \
              "com-tcp": 8013, \
              "status": 200 }'
```

Appendix C: Ein ganzer Tierladen mit SWAGGER.IO

https://petstore.swagger.io/

Mit SWAGGER kann man sehr gut REST-Schnittstellen *ohne Programmierung* austesten, allerdings braucht es dazu eine Spezifikation, die Sie quasi sich selbst zusammenstellen müssen. Das sieht dann (hoffentlich hübscher formatiert) z.B. so aus wie der Tierladen:

https://petstore.swagger.io/v2/swagger.json