Dokumentation CircleBits

Inhaltsverzeichnis

1. Funktionale Programmiermuster

DSL (Domain Specific Language): SQL-Code -> Realisieren der DB-Anbindung und Filter der relevanten Antwort zu den Fragen. Reminder-Funktion verwendet DSL

COO (Chain of Operations): -> bei Definition der Middleware

Meta-Programmierung: -> Die Funktion höherer Ordnung "Check Session" überprüft die Session und ruft die mitgegebene Funktion auf

2. Schnittstellen

Alle Pfade dieser Applikation (außer "POST /") erwarten den HTTP-REQUEST HEADER des Clients um die Session zu verifizieren!
Dieser wird unter "POST /" gesetzt. Bei Invalidierung oder bei einem Server-Neustart erhalten alle Nutzer einen 302 Redirect auf Moodle um die Session zu erneuern.

POST	/	Authentifizierung mithilfe der LTI gelieferten Daten. Es wird gerpüft ob der Nutzer bereits in der Datenbank ist. Falls nicht wird der Nutzer angelegt, ansonsten wird der Nutzer zurückgegeben.
GET	/	Indexseite, wird nach dem "POST /" aufgerufen und stellt die letzten (max. 20) Fragen+Antworten dar.

Alle folgenden Pfade sind mithilfe dem von LTI zur verfügung gestellten Feld "ROLE" geschützt.
Es wird die Rolle "Instructor" benötigt um Zugriff zu erhalten.
Instructor sind alle Administratoren des Moodle-Channels.

GET	/admin/	Stellt das Adminpanel (Website) bereit auf dem Antworten+Frage+Keywords angezeigt/hinzugefügt/gelöscht werden können.
GET	/admin/answers/	Zeigt alle Antworten in JSON Form an.
POST	/admin/answers/	Legt eine neue Antwort mit Keywords an bekommt "200 Successful" (erfolgreich) zurück.
GET	/admin/answers/:aid	Bekommt die Antwort mit der ID :aid als JSON zurück.
PUT	/admin/answers/:aid	Updated die Antwort mit der ID :aid gibt "200 ":aid (erfolgreich) zurück.
DELETE	/admin/answers/:aid	Löscht answer mit aid
GET	/reminder	Stellt das Reminder Panel zum erstellen und löschen von Remindern zur Verfügung.
GET	/reminder /reminders	Zeigt alle reminder
POST	/reminder /reminders	Legt einen neuen Reminder an
DELETE	/reminder/reminders/:rid	Löscht reminder mit rid

GET	/user/	Gibt den aktuellen Nutzer (abhängig von der Session) als JSON zurück.
GET	/user/logout/	Invalidiert die Session des aktuellen Nutzers.
GET	/user/:uid/	Gibt den Nutzer mit der ID :id als JSON zurück. Diese Methode verarbeitet ausschließlich die Daten, wenn die geforderte ID :uid mit der SessionID übereinstimmt.

Zu Administrationszwecken ist es der Rolle "Instructor" möglich über die ID :uid auf alle Profile LESEND zuzugreifen!

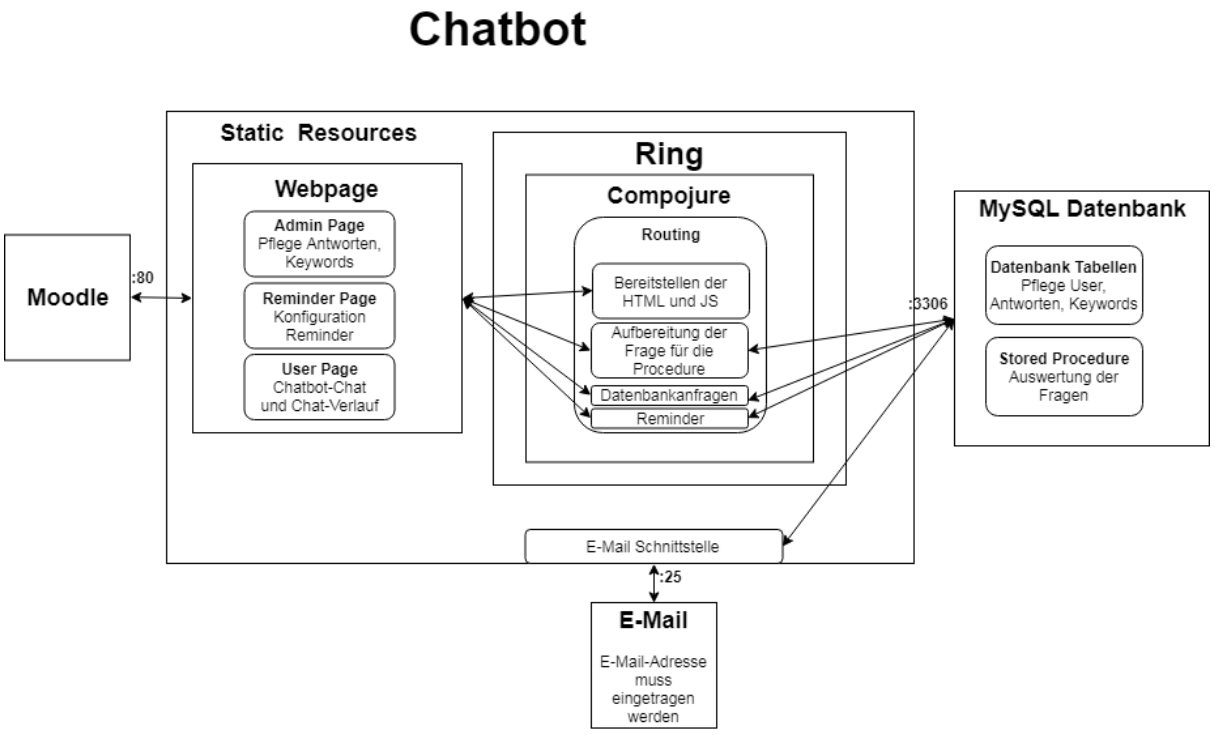
GET	/chats/	Gibt alle (max. 20) Chats des Nutzers als JSON zurück.
POST	/chats/	Erfordert den HTTP-BODY. Sendet die Frage an das Backend. Dort wird eine passende Antwort gesucht und mit dieser zusammen in der Datenbank abgespeichert. Bekommt "200" zurück bei erfolgreichem einfügen
GET	/chats/:uid/	Gibt alle Chats des Nutzers zurück, wenn dieser die gleiche ID :uid besitzt und somit der Besitzer ist.

Zu Administrationszwecken ist es der Rolle "Instructor" möglich über die ID :uid auf alle Profile LESEN zuzugreifen!

GET	/chats/:uid/:cid	Gibt die Frage+Antwort mit der ID :cid zurück, falls diese dem Nutzer mit der ID :uid gehört.

Zu Administrationszwecken ist es der Rolle "Instructor" möglich über die ID :uid auf den Chatverlauf mit der ID :cid LESEN zuzugreifen!

3. Architektur



4. Installation

4.1. DB Konfiguration

DB Shema import von Shema_Export_Circlebits.

Fallback:(**NUR**) bei Problemen mit dem Shema-Import Procedure Manuell Anlegen! Tables werden, falls nicht importiert, vom Service angelegt und müssen auch bei Fehlgeschlagenem Shema import, nicht manuell angelegt werden! Unsere Antworten sind nur mit dem Shema export enthalten.

Fragen können auch per Textdatei eingelesen werden. In der Datei DB.clj finden sich die nötigen Funktionen. Zum laden aus einer Datei reicht es aus die Variable pathToSTDAnswers mit dem Pfad zur Datei zu ersetzen und die Zeile ;(insertAnswerIfNotExists (first answerAndTrigger) (clojure.string/replace (second answerAndTrigger) #"(^[öÖäÄüÜßa-zA-Z0-9]*$)" in der Funktionsdefinition insertThoseAnswers zu entkommentieren.

Anleitung (**NUR** bei Problemen mit Shema import):
-Mit MySQL Workbench mit der Datenbank verbinden
-rechts klick auf die Stored Procedures

Folgenden Code einfügen:

```
CREATE PROCEDURE `GetAnswer`(IN IN_TRIGGER varchar(1000))
BEGIN

DECLARE count INT DEFAULT 0;
DECLARE best_aid INT DEFAULT 0;
DECLARE maxCount INT DEFAULT 0;

SET @trigger = CONCAT(CONCAT('(',IN_TRIGGER), ')');

SET @sql =CONCAT('SELECT COUNT(DISTINCT t1.aid) INTO @iterations FROM
ANSWERS as t1
JOIN KEYANSWER as t2
on t1.aid=t2.aid
JOIN KEYWORDS as t3
ON t2.kid=t3.kid
WHERE keyword in ',@trigger);

PREPARE stmt1 FROM @sql;
EXECUTE stmt1;
```

```
WHILE count < (SELECT @iterations) DO

SET @sql = CONCAT(CONCAT(CONCAT('SELECT DISTINCT(t1.aid) INTO @l_aid
FROM ANSWERS as t1
JOIN KEYANSWER as t2
on t1.aid=t2.aid
JOIN KEYWORDS as t3
ON t2.kid=t3.kid
WHERE t3.keyword in ',@trigger),
'ORDER BY t1.aid
LIMIT 1 OFFSET '), count);

PREPARE stmt1 FROM @sql;
EXECUTE stmt1;

SET @sql = CONCAT(CONCAT('(SELECT COUNT(*)
into @currentCount  FROM
(SELECT t4.aid FROM ANSWERS as t4
JOIN KEYANSWER as t5  on t4.aid=t5.aid
JOIN KEYWORDS as t6  ON t5.kid=t6.kid
WHERE keyword in ', @trigger) ,
' AND t4.aid IN (SELECT @l_aid))
as t)');

PREPARE stmt1 FROM @sql;
EXECUTE stmt1;

IF (SELECT @currentCount) > maxCount THEN
SET maxCount = (SELECT @currentCount);
SET best_aid =  (SELECT @l_aid);
END IF;

SET count = count + 1;
END WHILE;

IF best_aid = null OR best_aid = 0 THEN
SELECT * FROM ANSWERS WHERE aid = 1;
ELSE
SELECT * FROM ANSWERS WHERE aid = best_aid;
END IF;

END
```

4.2. Kontakt-Email hinterlegen

in der Datei email.clj müssen die Texte in <> eingetragen werden:
(def host <email_host>)
(def emailadresse <email_adresse>)
(def username <email_usernamet>)
(def emailpasswort <email_passwort>)
(def tlsEnable <true/false>)
(def port <port>) ;(Port 25 für tls)

4.3. Server starten

cd /home/<USER>/clojure-rest/
lein ring server-headless <PORT (default 3000)>

5. Prozessablauf

Klassifizierung der Benutzereingaben

1. User schickt Frage von Weboberfläche ab
2. Clojure Service empfängt Anfrage und bereitet sie auf
    a. Ring -> Compojure -> Clojure Funktion
    b. Entfernen aller Sonderzeichen
    c. toLowerCase und split an Leerzeichen
3. Senden der aufbereiteten Frage and die Datenbank
4. Stored Procedure wertet Frage aus
    a. Antwort mit den meisten übereinstimmenden Keywords wird gewählt
    b. Standard Antwort bei keiner Übereinstimmung
5. Datenbank -> Clojure -> Weboberfläche -> User
6. Clojure speichert den Chatverlauf in der Datenbank

6. Umgesetzte Anforderungen

6.1. E-Mail Kommunikation (Muss)

! Das E-Mail-Feature wird nach erstmaliger Anmeldung über Moodle aktiviert!

1. Der E-Mail Server legt einen Listener auf das Postfach der angegebenen E-Mail Adresse.
2. Bei Erhalt einer neuen Nachricht wird das Headers Feld nach der Matrikelnummer durchsucht und diese wird im Feld hsid der DB gesucht und geprüft.
3. Für den Fall, dass diese Matrikelnummer in unserer Datenbank verfügbar ist analysiert er die Nachricht und wertet sie aus. Falls sie nicht in der Datenbank verfügbar ist (der Nutzer noch nie über Moodle darauf zugegriffen hat) wird aus Sicherheitsgründen die Nachricht ignoriert und nicht verarbeitet.
4. Nach der Auswertung wird die Antwort gesendet. Der Nachrichtenverlauft wird, sowohl über die Website, als auch über die E-Mail im Profil gespeichert und auf der Weboberfläche ausgegeben.

6.2. Funktionale-Patterns (Muss)

1.DSL (Domain Specific Language): SQL-Code -> Realisieren der DB-Anbindung und Filter der relevanten Antwort zu den Fragen. Reminder-Funktion verwendet DSL

2.COO (Chain of Operations): -> bei Definition der Middleware

3.Meta-Programmierung: -> Die Funktion höherer Ordnung "Check Session" überprüft die Session und ruft die mitgegebene Funktion auf

6.3. Zustand für Benutzer aus Moodle Kurs (Muss)

Die Tabelle Users in der Datenbank ist mit den Benutzern des Moodle Kurses verknüpft.
Wird nach erstmaliger Anmeldung über Moodle aktiviert!

6.4. Dokumentation (Muss)

Dieses Dokument =)

6.5. Schnittstellenspezifikation (Muss)

Siehe 2. **Schnittstellen**.

6.6. Eine Frage zum Praxissemester beantworten (Muss)

Ist erfüllt, da der Service mehrere Fragen beantworten kann. Siehe 6.7..

6.7. Soll jeweils eine weitere Frage flexibel erkennen können (Soll)

Der Service kann, durch das Speichern der Antworten und die Auswertung in der Datenbank beliebig viele Fragen beantworten. Sie müssen durch die Admin-Page mit ihren Triggern(auslösenden Begriffen) definiert werden (20 wurden von uns definiert. 18 davon haben bei Tests funktioniert). Durch die manuelle Konfiguration ergibt sich ein höherer Aufwand, es erhöht jedoch die Flexibilität.

6.8. Web-Frontend (Soll)

Zur Konfiguration des Reminders/Erinnerungsfunktion, der Antworten stehen Konfigurationsseiten unter /reminder und /admin zur Verfügung. Der Pfad / stellt eine einfache Chatseite mit Chatverlauf bereit.

6.9. Kommunikationsablauf/-verlauf(Soll)

Der Chatverlauf wird für jeden angemeldeten Nutzer erfasst und ihm auf der Seite angezeigt. Auch Fragen, die per Mail gestellt wurden, tauchen im Chatverlauf auf.

6.10. Erinnerungsfunktion (Soll)

Da eine generische Lösung erstellt wurde, wurden keine Reminder erstellt. Da beliebige definiert und gelöscht werden können schien das nicht nötig. Die Anforderung sollte dennoch erfüllt sein.

6.11. Erinnerungsfunktion Generisch (Soll)

Über den Pfad /reminder können Reminder/Erinnerungen beliebig definiert und gelöscht werden. Es kann ein Zeitpunkt, eine Nachricht und ein Betreff definiert werden. Der Zeitpunkt muss dabei im cron-Format definiert werden (unter https://crontab.guru kann eine Bschreibung angezeigt werden). Die Dokumentation der verwendeten Library kann unter https://github.com/clojurewerkz/quartzite.docs eingesehen werden.

6.12. Den Aufbau des Micro-Service erläutern und die verwendeten Design Patterns für funktionale Programmierung aufzeigen (Soll)

Siehe 3.**Architektur** und Anforderung 6.2 und 1.**Funktionale Programmiermuster**

6.13. Mechanismus, mit dem der entwickelte Micro-Service Benutzereingaben klassifiziert, beschreiben (Soll)

5.**Prozessablauf**