JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von

Alexander Pabinger, BSc

Daniel Putschögl, BSc

Angefertigt am

Institut für Wirtschaftsinformatik – Communications Engineering

PROJEKT-DOKUMENTATION

Implementierung eines Moodle Plugins zum File-Upload

Betreuer / Betreuerin

Dr. Stefan Oppl

März 2020

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

1. Projektbeschreibung

Das Projekt wurde im Zuge des Praktikums Business Engineering & Management am Institut für Wirtschaftsinformatik – Communications Engineering an der Johannes-Kepler-Universität durchgeführt. Die Durchführung des Projektes erfolgte im Wintersemester 2019/20. Die Projektbetreuung erfolgt durch Stefan Oppl.

1.1. Kurzbeschreibung

Im Zuge dieses Projektes wird die bestehende Infrastruktur am Institut für Wirtschaftsinformatik – Communication Engineering zur Generierung von PDF-Dokumenten aus Markdown um eine automatisierte Content-Bereitstellung auf der Lernplattform Moodle erweitert. Hierzu wird der über Gitlab angestoßene PDF-Build-Prozess um einen Delivery-Schritt erweitert, wobei das erstellte PDF über eine Schnittstelle auf Moodle hochgeladen wird.

1.2. Vorarbeit und vorhandene Dokumente

Am Institut für Wirtschaftsinformatik – Communications Engineering ist momentan ein System vorhanden, welches aus Markdown PDF-Dokumente generiert werden. Markdown ist eine einfache, einsteigerfreundliche Markup Sprache mit dem Ziel, schon die Ausgangsform leicht lesbar zu machen. In einem vorherigen Projekt „Publishing Toolchain mit Markdown“ wurde dieses System erweitert, um eine hohe Qualität dieser Dokumente sicherzustellen, PDF-Dateien daraus zu erzeugen und HTML-Exports für den Import zu generieren. Das Tool pandoc bietet dem User die Möglichkeit auf Basis verschiedener Templates aus dem Ausgangsformat diverse Zielformate zu generieren. Mithilfe von pandoc wird am Institut aus dem Markdown-File ein PDF-Dokument generiert.

Docker ist eine Virtualisierungsplattform die es ermöglicht, Software in sogenannten Containern auf Betriebssystemebene virtualisiert bereitzustellen. Alle Container werden von einem einzigen Betriebssystem-Kernel betrieben und sind somit leichter als virtuelle Maschinen. GitLab ist ein webbasiertes DevOps Lifecycle-Tool, das einen Git-Repository-Manager, Issue-Tracking und CI/CD-Pipeline Funktionen bietet. Auf Basis der Gitlab Continuous Integration Services wird also ein Docker Container gestartet, in welchem die PDF-Generierung mithilfe der bereits genannten Tools durchgeführt wird. Gitlab selbst wird vom Institut auf einem CentOs-Server gehostet und unter https://gitlab.ce.jku.at/ zur Verfügung gestellt.

Vorhandene Dokumente:

- Technische Dokumentation des Projektes „Publishing Toolchain mit Markdown“
- Moodle API Dokumentation

1.3. Ziele

Ziel dieses Projektes ist es, auf das vorherige Projekt aufzubauen und dieses zu erweitern. Durch einen Knopfdruck ist es möglich, eine aus der Gitlab-Pipeline erzeugte PDF-Datei zu einem Kurs in Moodle hinzuzufügen.

2. Implementierung

2.1. Moodle und Postgres in Docker

Zum Testen von Moodle in der Version 3.7 wurden Moodle und Postgres als darunter liegende Datenbank über Docker Container bereitgestellt. Hierzu wird mittels nachfolgendem Dockerfile das Moodle Image erstellt.

```
FROM ubuntu

ARG DEBIAN_FRONTEND=noninteractive

RUN apt-get update &&\
   apt-get install -y git apache2 php php-fpm php-curl php-zip php-pgsql php-pspell php-gd
         php-intl php-xml php-xmlrpc php-ldap php-soap php-mbstring libapache2-mod-php graphviz
         aspell ghostscript clamav acl sudo &&\
   apt-get clean

RUN a2enmod proxy_fcgi setenvif &&\
   a2enconf php7.2-fpm

#COPY moodle /var/www/html/moodle
RUN mkdir /var/www/html/moodle &&\
   chown -R www-data /var/www/html/moodle &&\
   chmod -R 755 /var/www/html/moodle

RUN mkdir /var/www/moodledata &&\
   chown -R www-data /var/www/moodledata &&\
   chmod -R 777 /var/www/moodledata

COPY entrypoint.sh /var/entrypoint.sh
RUN chmod 755 /var/entrypoint.sh
ENTRYPOINT ["/var/entrypoint.sh"]

EXPOSE 80
CMD apachectl -D FOREGROUND
```

Mithilfe des Dockerfiles wird ein Ubuntu Image erstellt und die nötigen Daten für die Moodle-Installation werden in das Image kopiert. Da zum Zeitpunkt der Moodle-Installation die Datenbank-Instanz bereits laufen muss, kann Moodle hier zum Zeitpunkt der Image-Erstellung nicht installiert werden. Dies passiert beim ersten Container-Startup mithilfe folgendes Scriptes:

```
#!/bin/bash
set -e

if [ ! -f /var/www/html/moodle/config.php ]; then
sudo -u www-data /usr/bin/php /var/www/html/moodle/admin/cli/install.php --lang=en --
dbtype=$DB_TYPE --dbhost=$DB_HOST --dbname=$DB_NAME --dbuser=$DB_USER --
dbpass=$DB_PASS --dbport=$DB_PORT --non-interactive --agree-license --allow-unstable --
adminuser=admin --adminpass=password --wwwroot=http://$WWW_ROOT/moodle --
dataroot=$DATA_ROOT --fullname=moodle --shortname=moodle
fi

chown -R www-data /var/www

exec "$@"
```

Die beiden Container für Postgres und Moodle werden mit folgendem docker-compose.yaml gestartet:

```
version: '2.1'

services:
 db:
  restart: always
  image: postgres:11
  volumes:
   - ./postgresdata:/var/lib/postgressql/data/pgdata
  container_name: db
  environment:
   - POSTGRES_USER=moodleuser
   - POSTGRES_PASSWORD=password
   - POSTGRES_DB=moodle
   - PGDATA=/var/lib/postgressql/data/pgdata
  healthcheck:
     test: ["CMD-SHELL", "pg_isready -U moodleuser -d moodle"]
     timeout: 5s
     retries: 5
  networks:
     moodle_network:
        ipv4_address: 172.28.0.2
 moodle:
  build: ./moodle-docker
  volumes:
   - ./moodle-docker/moodledata:/var/www/moodledata
   - ./moodle-docker/moodle:/var/www/html/moodle
  container_name: moodle
  depends_on:
     db:
        condition: service_healthy
```

```
    environment:
      - DB_TYPE=pgsql
      - DB_HOST=db
      - DB_PORT=5432
      - DB_NAME=moodle
      - DB_USER=moodleuser
      - DB_PASS=password
      - DATA_ROOT=/var/www/moodledata
      - WWW_ROOT=172.28.0.3
    networks:
      moodle_network:
        ipv4_address: 172.28.0.3

networks:
  moodle_network:
    ipam:
      driver: default
      config:
        - subnet: 172.28.0.0/16
```

Die Daten werden jeweils im Ordner postgresdata bzw. moodledata persistiert. Zusätzlich werden die Umgebungsvariablen für Moodle gesetzt, welche bei der Installation benötigt werden. Moodle wird erst hochgefahren und installiert, wenn die Postgres-Instanz läuft.

2.2. Plugin „uploadfiletocourse“ Beschreibung

Nach einer ausgiebigen Internetrecherche konnte festgestellt werden, dass kein Webservice für den File-Upload zu einem Kurs über eine REST-Schnittstelle für Moodle existiert. Aus diesem Grund, wurde ein eigenes Plugin implementiert, welches in diesem Kapitel vorgestellt wird.

Unter folgendem Link ist ein Template zur Erstellung eines Plugins verfügbar: https://github.com/moodlehq/moodle-local_wstemplate

Aus dem Template ergibt sich folgende Filestruktur:

client: client.php

Zum Testen des Webservices

db: services.php

Hier wird der Webservice zur automatischen Erkennung in Moodle definiert

externallib.php:

Hier wird die Hauptfunktionalität implementiert

version.php:

Dieses File dient zur Versionsverwaltung

2.2.1. db: services.php

Dieser Schritt ist optional. Man kann einen Dienst mit beliebigen Web-Service-Funktionen vorkonfigurieren, so dass der Moodle-Administrator dies nicht tun muss.

2.2.1.1. Service deklarieren

In /local/myplugin/db/services.php hinzufügen:

```
<?php
$services = array(
   'uploadfiletocourseplugininservice' => array(                                    //the name of the web
service
      'functions' => array ('custom_upload_file_to_course'), //web service functions of this service
      'requiredcapability' => '',              //if set, the web service user need this capability to access
      //any function of this service. For example: 'some/capability:specified'
      'restrictedusers' =>0,                                    //if enabled, the Moodle administrator must
link some user to this service
      //into the administration
      'enabled'=>1,                                          //if enabled, the service can be reachable on
a default installation
   )
);
```

2.2.1.2. Web Service Funktionen deklarieren

```
$functions = array(
   'custom_upload_file_to_course' => array(        //web service function name
      'classname'   => 'custom_upload_file_to_course',  //class containing the external function
      'methodname'  => 'upload_file_to_course',          //external function name
      'classpath'   => 'local/uploadfiletocourseplugin/externallib.php',  //file containing the
class/external function
      'description' => 'Uploads a file to a specific course.',    //human readable description of the web
service function
      'type'        => 'write',                  //database rights of the web service function (read, write)
      'ajax' => true,        // is the service available to 'internal' ajax calls.
      'services' => array(MOODLE_OFFICIAL_MOBILE_SERVICE)    // Optional, only available for
Moodle 3.1 onwards. List of built-in services (by shortname) where the function will be included.
Services created manually via the Moodle interface are not supported.
   ),
);
```

2.2.2. externallib.php

Um eine Webservice-Funktion erstellen zu können, wird eine Klasse (custom_upload_file_to_course) erstellt, welche drei verschiedene Methoden enthält. Im Folgenden werden diese Methoden näher betrachtet:

2.2.2.1. upload_file_to_course_parameters ()

Diese Funktion dient zur Übergabe der Parameter für die Hauptmethode. In diesem Fall werden zwei Parameter angegeben:

- Courseid (Typ: int, Pflichtfeld): Datenbank ID des Kurses, zu welchem die Datei hochgeladen werden soll.
- Filename (Typ: file, Optional): Dateiname des Files, welches hochgeladen werden soll. Falls dieser nicht angegeben ist, wird der Name aus der Datei übernommen.

```
public static function upload_file_to_course_parameters()
{
   return new external_function_parameters(
      array(
         'courseid' => new external_value(PARAM_INT, 'course id'),
         'filename' => new external_value(PARAM_FILE, 'file name', VALUE_OPTIONAL),
      )
   );
}
```

Erklärung der Parameterdefinitionen:

- PARAM_INT: integers only, use when expecting only numbers.
- PARAM_FILE: safe file name, all dangerous chars are stripped, protects against XSS, SQL injections and directory traversals

2.2.2.2. upload_file_to_course_returns ()

In dieser Methode wird der Rückgabewert der Hauptfunktion deklariert. In unserem Fall geben wir nur einen Text zurück, ob der Upload erfolgreich war.

```
public static function upload_file_to_course_returns()
{
   return new external_value(PARAM_TEXT, 'Status message');
}
```

2.2.2.3. upload_file_to_course ()

Diese Methode enthält die Hauptfunktionalität unseres Plugins. Wie bereits oben erwähnt, werden als Parameter die eindeutige Kurs ID und optional der Dateiname angegeben. Zusätzlich werden zwei Hilfsmethoden dazu verwendet, um benötigte Instanzen zu erzeugen.

```
public static function upload_file_to_course($courseid, $filename = "")
{
   global $DB;
   // We have to create a random item_id for the file
   $random_itemid = rand(100000000, 999999999);
   // If random item_id already exists, create a new one
   $file = $DB->get_record('files', array('itemid' => $random_itemid), '*');
```

```
    while ($file) {
        $random_itemid = rand(100000000, 999999999);
        $file = $DB->get_record('files', array('itemid' => $random_itemid), '*');
    }
    // Main upload functionality provided in repository API
    $resp = (new repository_upload(4))->process_upload($filename, -1, $types = '*', $savepath = '/',
$random_itemid);
    if (empty($filename)){
        $filename = $resp['file'];
    }
    try {
        $course = $DB->get_record('course', array('id' => $courseid), '*', MUST_EXIST);
    } catch (dml_exception $e) {
        throw new dml_exception('Course does not exist. Please try another ID');
    }
    $newcm = self::coursemodule_helper($course);
    $transaction = $DB->start_delegated_transaction();
    $coursemodule_id = add_course_module($newcm);
    $data = self::resource_helper($filename, $courseid, $random_itemid, $coursemodule_id);
    $returnfromfunc = resource_add_instance($data, NULL);
    $DB->set_field('course_modules', 'instance', $returnfromfunc, array('id' => $coursemodule_id));
    $modcontext = context_module::instance($coursemodule_id);
    $sectionid = course_add_cm_to_section($course, $coursemodule_id, 0);
    $transaction->allow_commit();
    return 'Successfully uploaded file ' .  $filename . ' to course: ' . $course->fullname;
}
//
// Helper -----------------------------------------------------
//
/**
 * Helper to create a resource instance
 *
 * @param $filename
 * @param $courseid
 * @param $random_itemid
 * @param $coursemodule_id
 * @return stdClass
 */
private static function resource_helper($filename, $courseid, $random_itemid, $coursemodule_id)
{
    ...
}
/**
 * Helper to create a coursemodule instance
 *
 * @param $course
 * @return stdClass
```

```
 */
private static function coursemodule_helper($course)
{
    ...
}
```

3. Nutzung des Plugins

Die folgenden Schritte müssen zur erfolgreichen Nutzung des Plugins ausgeführt werden.

3.1. Plugin herunterladen

Die einfachere Variante zur Verwendung des Plugins ist, den ZIP-Ordner aus dem Anhang zu nutzen.

Falls dieser nicht vorhanden ist, kann das Plugin als ZIP von Github heruntergeladen werden. Hier ist zu beachten, dass nach dem Download die .git Datei im Ordner gelöscht werden muss, um das Plugin über die Benutzeroberfläche installieren zu können.

Github Link zum Plugin: https://github.com/AlexDelPab/moodle_plugin_uploadfiletocourse.git

3.2. Plugin installieren

Es existieren verschiedene Varianten für die Installation des Plugins. In diesem Fall wird nur die einfachere, d.h. Installation über die Benutzeroberfläche, beschrieben.

Unter ***Site administration > Plugins > Install plugins***

1. ZIP-Datei auswählen

2. „Install plugin from the ZIP file“ drücken

3. Validation successful: „Fortfahren“

4. Plugins check: „Upgrade Moodle database now“ drücken

3.3. Webservice aktivieren

Unter ***Site administration > Plugins > Web services (Overview)*** folgende Schritte ausführen:

1. Enable web services: Yes

2. Enable protocols: rest

3. Create a specific user: Hier einen Benutzer anlegen, der für den Webservice genutzt werden soll

3.4. Token für Plugin erzeugen

Unter ***Site administration > Plugins > Web services > Manage tokens > Add*** kann nun ein Token für dieses Plugin und dem zuvor erstellen Web Service Nutzer angelegt werden.

Create token:

1. Webservice User auswählen

2. Service: uploadfiletocourse**p**luginservice auswählen

3. Speichern

Dieses Token benötigen wir für den Client zur Ansprache des Webservice.

4. Beispiel Clients

4.1. Postman

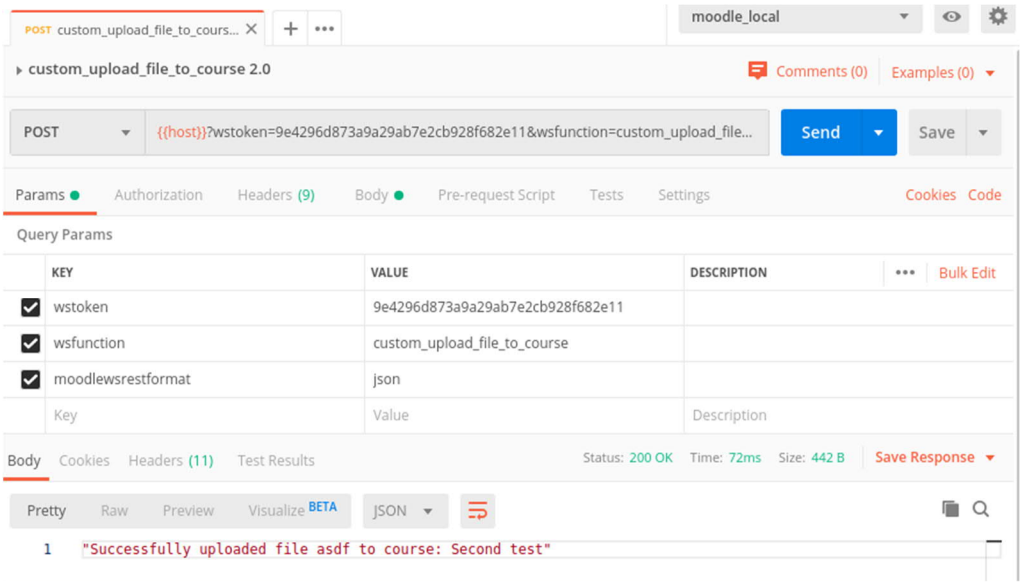

Abbildung 2: Webservice Aufruf in Postman (Params)

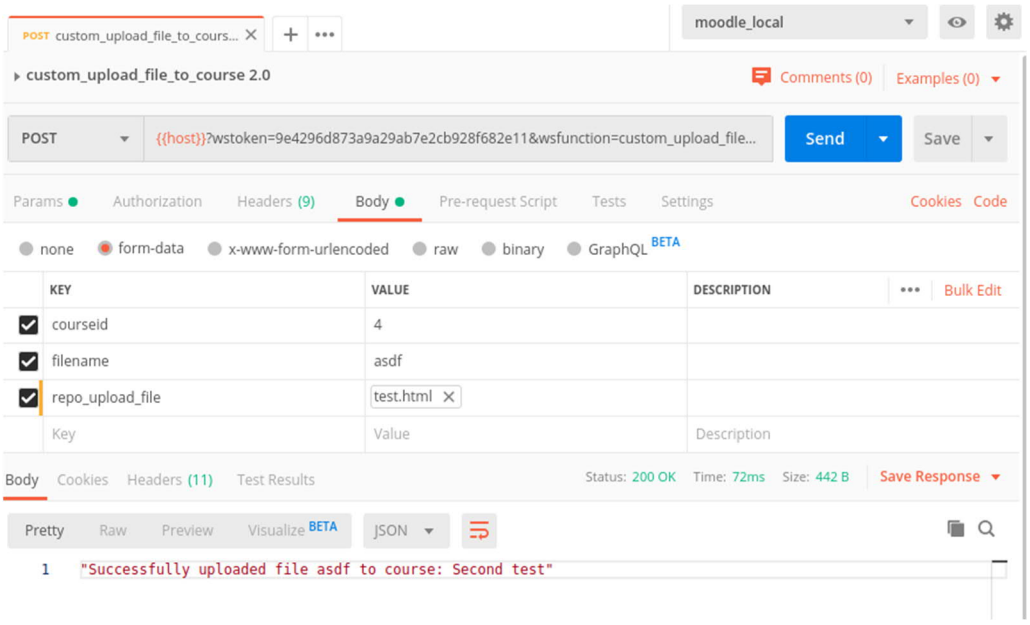

Abbildung 1: Webservice Aufruf in Postman (Body)

Im Folgenden wurden zwei verschiedene Webservice-Aufrufe aus Postman exportiert, um beispielhaft den Aufruf darzustellen.

4.2. PHP – cURL

```
<?php

$curl = curl_init();

curl_setopt_array($curl, array(
 CURLOPT_URL =>
"http://172.28.0.3/moodle/webservice/rest/server.php?wstoken=9e4296d873a9a29ab7e2cb928f682e
11&wsfunction=custom_upload_file_to_course&moodlewsrestformat=json",
 CURLOPT_RETURNTRANSFER => true,
 CURLOPT_ENCODING => "",
 CURLOPT_MAXREDIRS => 10,
 CURLOPT_TIMEOUT => 0,
 CURLOPT_FOLLOWLOCATION => true,
 CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
 CURLOPT_CUSTOMREQUEST => "POST",
 CURLOPT_POSTFIELDS => array('courseid' => '4','filename' => 'Test.html','repo_upload_file'=> new
CURLFILE('/home/alex/Documents/test.html')),
 CURLOPT_HTTPHEADER => array(
   "Content-Type: multipart/form-data; boundary=--------------------------
849689444351296439937880"
 ),
));

$response = curl_exec($curl);

curl_close($curl);
echo $response;
```

4.3. JavaScript – jQuery

```
var form = new FormData();
form.append("courseid", "4");
form.append("filename", "Test.html");
form.append("repo_upload_file", fileInput.files[0], "test.html");

var settings = {
 "url":
"http://172.28.0.3/moodle/webservice/rest/server.php?wstoken=9e4296d873a9a29ab7e2cb928f682e
11&wsfunction=custom_upload_file_to_course&moodlewsrestformat=json",
 "method": "POST",
 "timeout": 0,
 "headers": {
  "Content-Type": "multipart/form-data; boundary=--------------------------
849689444351296439937880"
 },
 "processData": false,
 "mimeType": "multipart/form-data",
 "contentType": false,
 "data": form
};

$.ajax(settings).done(function (response) {
 console.log(response);
});
```

4.4. Bash Script (.sh)

```
curl --request POST \
 --url 'https://e-bfi-
ooe.at/webservice/rest/server.php?wstoken=9010dc558b636bf3be9976d872caf003&wsfunction=custo
m_upload_file_to_course&moodlewsrestformat=json' \
 --header 'cache-control: no-cache' \
 --form 'courseid=4' \
 --form 'filename=Test.html' \
 --form 'repo_upload_file=@./test.html'
```

5. Projektplan

Das Praktikum erstreckte sich über das Wintersemester 2019/20 und wurde in vier Phasen geplant, welche in der folgenden Tabelle näher beschrieben werden. Projektphase 1 und 2 verliefen teilweise parallel, da die Erstellung des Projektplans gleichzeitig mit der Einarbeitung in die Dokumentationen erfolgte. In Summe wurden **106 Stunden pro Person** als Aufwand geschätzt, welche wie folgt aufschlüsseln.

Phase	Von	Bis	Beschreibung	Meilenstein	Aufwands-schätzung in h / p. P-
1	**02.07.**	**31.10.**	**Vorbereitung**	**Projektvorbereitung**	**10**
1.1			Projektziel definieren		4
1.2			Projektplan erstellen		5
1.3			Ablauf festlegen		1
2	**01.11.**	**30.11.**	**Einarbeitung**	**Projekteinarbeitung**	**20**
2.1			Moodle (REST API, …)		5
2.2			Gitlab (CI/CD, …)		5
2.3			Docker (Virtualization, …)		5
2.4			Sonstiges (pandoc, PostgreSQL, Postman, …)		5
3	**01.12.**	**31.12.**	**Entwicklung**	**Projektdurchführung**	**75**
3.1			Containerisierung von Moodle und PostgreSQL (inklusive DB Erstellung) durch Docker		10
3.2			Erstellung/Ansprache der Moodle REST API zum Empfang der PDF-Datei		30
3.3			API Test mit Postman		5
3.4			Automatisierung des Uploads über die Gitlab-Pipeline		30
4	**31.01.**	**31.01.**	**Abgabe**	**Projektabschluss**	**1**

Tabelle 1: Projektplanung

6. Reflexion

Die Meilensteine Projektvorbereitung und Projekteinarbeitung verliefen wie geplant. Das Einlesen in die Moodle-Dokumentation war teilweise herausfordernd, da einerseits die Dokumentation sehr umfangreich ist und andererseits keine Beispiele beinhaltet. Die Einarbeitung in Docker, Gitlab und co. erfolgte teilweise Hands-on durch Projektphase 3. Die Containerisierung war von der Aufwandsschätzung zu niedrig, da diese „teilweise“ manuell durchgeführt wurde und hier die Erfahrung im Umgang mit Docker und Docker-Compose fehlte. Ein weiteres Problem bei der Containerisierung waren außerdem zwei unterschiedliche Betriebssysteme (Windows und Ubuntu), bis letztendlich einheitlich auf Ubuntu entwickelt wurde. Die Erstellung der Moodle Schnittstelle war aufgrund der schlechten Dokumentation herausfordernd und man musste sich genau in den Source Code von Moodle einlesen. Darüber hinaus hat sich der Aufwand deutlich dadurch erhöht, dass Moodle nur eine sehr rudimentäre REST-Schnittstelle zur Verfügung stellt und viele grundlegende Funktionen, wie der Fileupload, nicht existieren. Deshalb musste ein eigenes Plugin implementiert werden, welches diese Funktionalität bereitstellt. Das Testen der API mit Postman und die Automatisierung des Uploads über die Gitlab-Pipeline erfolgten ohne Probleme.

GitHub-Repository mit allen erstellten Inhalten: https://github.com/danielputschoegl/busempr