



UNIVERSITÄT OSNABRÜCK

INSTITUT FÜR INFORMATIK  
AG SOFTWARE ENGINEERING

*Masterarbeit*

# **Anbindung von Messaging-Systemen an Lernmanagementsysteme (am Beispiel von Stud.IP und Matrix)**

Manuel Schwarz

August 2021

Erstgutachter: Dr. Tobias Thelen  
Zweitgutachterin: Prof. Dr. Elke Pulvermüller



## **Zusammenfassung**

Die vorliegende Arbeit entstand in der Arbeitsgruppe Software Engineering an der Universität Osnabrück im Bereich Webentwicklung und befasst sich mit der Anbindung von Messaging-Systemen an Lernmanagementsysteme. Kern der Arbeit ist die Erstellung und sinnvolle Umsetzung eines Anforderungskatalogs für das oben genannte Thema am Beispiel der Messaging-Software Matrix und der Lernplattform Stud.IP.

Einleitend wird eine kurze Motivation des Themas gegeben. Daraufhin folgt ein Grundlagenkapitel mit allen für diese Arbeit relevanten Hintergrundinformationen. Die Anforderungsanalyse bildet das nächste Kapitel, in dem die Erhebung sowie alle daraus entstandenen Anforderungen aufgelistet und beschrieben werden. Anschließend wird die Umsetzung der herausgearbeiteten Anforderungen im Implementationskapitel dargestellt.

Abschließend endet die Arbeit mit einer kurzen Evaluation sowie einem Ausblick, der sich mit möglichen Verbesserungen und Erweiterungen für den Praxiseinsatz beschäftigt.



# Inhaltsverzeichnis

<b>1</b>	<b>Motivation</b>	<b>1</b>
<b>2</b>	<b>Hintergrund</b>	<b>3</b>
2.1	Lernmanagementsysteme . . . . .	3
2.2	Stud.IP . . . . .	4
2.2.1	Stud.IP Plugins . . . . .	5
2.2.2	JSON:API . . . . .	6
2.2.3	Blubber-Chat . . . . .	7
2.2.4	Notifications . . . . .	8
2.3	Anbindung von Fremdsystemen an Stud.IP . . . . .	9
2.3.1	Meetings Plugin . . . . .	9
2.3.2	Etherpad Plugin . . . . .	11
2.3.3	Moodle-Connect Plugin . . . . .	13
2.4	Matrix-Messenger . . . . .	14
2.4.1	Matrix Schnittstellen . . . . .	15
2.4.2	Matrix Client . . . . .	15
<b>3</b>	<b>Anforderungsanalyse</b>	<b>17</b>
3.1	Umfrage zur Anforderungsanalyse . . . . .	17
3.2	Grundsatzentscheidung . . . . .	19
3.3	Anforderungen . . . . .	19
3.3.1	Veranstaltungen auf Räume abbilden . . . . .	20
3.3.2	Übernahme von Rollen, Rechten und Status . . . . .	20
3.3.3	kein zusätzliches Login . . . . .	21
3.3.4	Benachrichtigungsfunktion . . . . .	21
3.3.5	Transparenz . . . . .	21
3.3.6	Direktnachrichten . . . . .	22
3.3.7	mehrere Räume pro Kurs . . . . .	22
3.3.8	Kompatibilität von Nachrichten in Blubber und Matrix . . . . .	22
3.3.9	Übernahme des Stud.IP Profilbildes . . . . .	22
3.3.10	Stautsindikator in Stud.IP . . . . .	22
3.3.11	Anbindung von Matrix durch ein Stud.IP-Plugin . . . . .	23
3.3.12	Robustheit, Ausfallsicherheit . . . . .	23

<b>4</b>	<b>Implementation</b>	<b>25</b>
4.1	Veranstaltungen auf Räume abbilden . . . . .	25
4.2	Übernahme von Rollen, Rechten und Status . . . . .	25
4.3	kein zusätzliches Login . . . . .	26
4.4	Benachrichtigungsfunktion . . . . .	26
4.5	Transparenz . . . . .	26
4.6	Direktnachrichten . . . . .	27
4.7	mehrere Räume pro Kurs . . . . .	27
4.8	Kompatibilität von Nachrichten in Blubber und Matrix . . . . .	27
4.9	Übernahme des Stud.IP Profilbildes . . . . .	27
4.10	Stautsindikator in Stud.IP . . . . .	27
4.11	Anbindung von Matrix durch ein Stud.IP-Plugin . . . . .	27
4.12	Robustheit, Ausfallsicherheit . . . . .	28
<b>5</b>	<b>Evaluation</b>	<b>31</b>
5.1	Chancen . . . . .	31
5.2	Probleme . . . . .	31
<b>6</b>	<b>Ausblick</b>	<b>33</b>
<b>A</b>	<b>Umfrageergebnisse zur Anforderungsanalyse der Stud.IP-Matrix-Integration</b>	<b>37</b>

# Abbildungsverzeichnis

2.1	LMS Architektur . . . . .	4
2.2	Stud.IP Pluginverwaltung . . . . .	5
2.3	Stud.IP JSON:API Ablaufdiagramm . . . . .	7
2.4	Stud.IP: Globaler Blubber . . . . .	8
2.5	Meetings-Plugin: Dialog bei Raumerstellung . . . . .	10
2.6	Etherpad: Einbettung mit <code>iframe</code> . . . . .	12
2.7	Dozentensicht: Moodle-Connect Plugin . . . . .	13
3.1	Umfrage: Anforderungsanalyse . . . . .	18
3.2	Umfrage: Teilnehmendenverteilung . . . . .	18
4.1	Matrix: Adminkonfiguration . . . . .	29





# Kapitel 1

## Motivation

Insbesondere motiviert ist diese Arbeit durch die fast vollständig digital durchgeführten „Corona“-Semester in den letzten 1.5 Jahren. Die Digitalisierung der Lehre ist schlagartig und stärker als zuvor in den öffentlichen Fokus gelangt und hat zusätzlich an Relevanz gewonnen, da man plötzlich darauf angewiesen war und weiterhin ist.

Lehren und Lernen im digitalen Zeitalter spielt eine immer zentralere Rolle und zeigt neue, bisher vielleicht nicht gesehene Chancen auf.

Moderne Kommunikationsmittel und -wege sollen studiumsunterstützend eingesetzt werden. Nach Möglichkeit sollen aktuelle und moderne Kommunikationswerkzeuge genutzt werden, um Studierenden eine möglichst niedrige Einstiegsschwelle bei Fragen oder Unklarheiten zu bieten. Als Betreiber eines Lernmanagementsystems befindet sich die Hochschule in einem fortwährenden Prozess der Weiterentwicklung und Anpassung zur Vermittlung von Informationen.

Wie studiert man heute? Mit dieser und weiteren Fragen befasste sich eine bundesweite Studie zum Thema der Digitalisierung in der Hochschullehre im Jahr 2020.[22]

Unter anderem wurde herausgefunden, dass Studierende die Digitalisierung fast aller für die Organisation von Studium und Lehre relevanten Prozesse mehrheitlich als wichtig betrachten. Weiterhin sind die als am wichtigsten bewerteten Kommunikationswerkzeuge die Hochschul- sowie Privat-E-Mail und der Chat [22][S.22]. Plattforminterne Nachrichten und Forum sind eher im unteren Wichtigkeitsmittelfeld anzutreffen.

Ein großer Punkt, der in der Studie herausgestellt wird, ist, dass fast die Hälfte der Studierenden private Plattformen für Studienzwecke nutzt, obwohl sich eine deutliche Mehrheit wünscht, dass dies nicht der Fall wäre.

Vor diesem Hintergrund wirkt ein hochschuleigener Chat-Server (Matrix) inklusive selbst gehostetem Matrix-Client (Element) mit einer Anbindung an Stud.IP wie ein Schritt in die richtige Richtung. Da bereits viel der alltäglichen Kommunikation über mobile Endgeräte stattfindet, könnten somit sogar Synergieeffekte auftreten und man spart sich den Umweg über die Lernplattform.



# Kapitel 2

## Hintergrund

Das folgende Kapitel befasst sich knapp mit den Grundlagen, die dieser Arbeit zu Grunde liegen. Begonnen bei Lernmanagementsystemen und Messaging-Systemen, hin zu den konkreten Implementationen Stud.IP und Matrix sowie Beispielen für die Anbindung anderer Fremdsysteme an die Lernplattform Stud.IP.

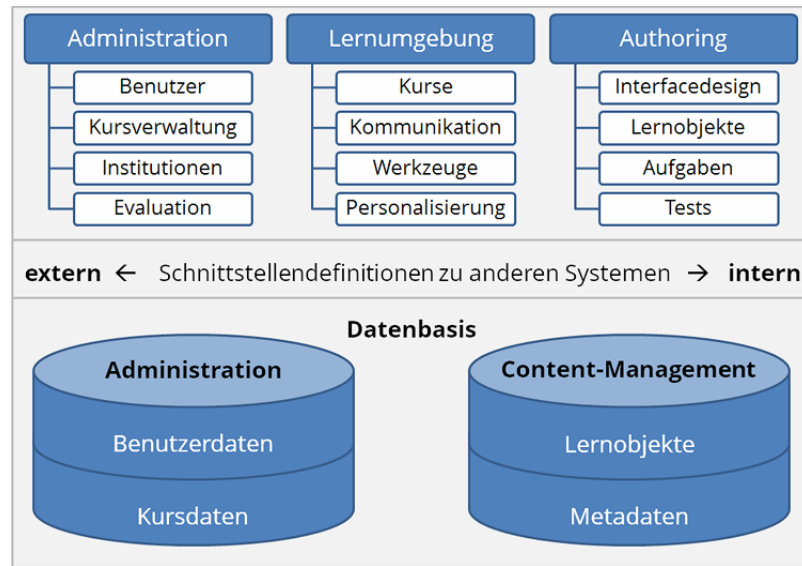
### 2.1 Lernmanagementsysteme

Lernmanagementsysteme, im Folgenden mit LMS abgekürzt, dienen der Unterstützung von Lehr- und Lernprozessen im Kontext von E-Learning sowie der Verwaltung von Lehrmaterialien und Nutzerdaten. Diese Online-Lernplattformen bilden an vielen Hochschulen aber auch anderen Institutionen die Basis einer E-Learning-Infrastruktur.

Die Hauptfunktion, die ein LMS in der Regel erfüllt, ist die Organisation des Lehrbetriebs, insbesondere das Anlegen von Veranstaltungen, die Bereitstellung von Lerninhalten und die Kommunikation zwischen Lehrenden und Lernenden.

Definition nach Schulmeister [14, S. 10]: Als Lernplattformen oder Learning Management Systeme werden im Unterschied zu bloßen Kollektionen von Lehrskripten oder Hypertext-Sammlungen auf Webservern solche Software-Systeme bezeichnet, die über folgende Funktionen verfügen:

- Eine Benutzerverwaltung
- Eine Kursverwaltung (Inhalte, Dateien)
- Ein Rollen- und Rechtesystem
- Kommunikationsmethoden (Chat, Foren) und Werkzeuge für das Lernen (Whiteboard, Notizbuch, Annotationen, Kalender etc.)
- Eine webbasierte Darstellung der Kursinhalte, Lernobjekte und Medien.



**Abbildung 2.1:** LMS Architektur [14, S. 11]

Die Abbildung 2.1 zeigt die Architektur eines idealtypischen LMS. Auf der Ebene der Datenbasis bildet meist ein Content-Management-System die Grundlage der Datenverwaltung. Neben Personen- und Kursdaten, können ebenso Lerninhalte archiviert, verteilt und wiederverwendet werden.

Darauf aufbauend ist es möglich je nach Bedarf interne sowie externe Schnittstellen zu anderen Systemen zu definieren.

Auf höchster Abstraktionsebene stellt das Diagramm drei Hauptkategorien der Nutzung dar. Zunächst sollte es eine administrative Ebene mit einer Nutzer-, Institutions-, und Kursverwaltung sowie der Option für Evaluationen geben. Die zweite Kategorie repräsentiert eine Lernumgebung mit ihren Werkzeugen und Kommunikationswegen. Der letzte Block beschreibt das Authoring, dass die konkreten Lernobjekte, Aufgaben und Tests umfasst.

## 2.2 Stud.IP

Stud.IP ist eine Open Source Implementation eines LMS, dass Anfang der 2000er Jahre entwickelt wurde. Mit ihren offenen Schnittstellen erlaubt diese digitale Lernplattform die Integration von externen Systemen und Anwendungen zum Einsatz in der digitalen Lehre. Neben Hochschulen gehören ebenfalls Schulen, Unternehmen, Verbände und Behörden zu den Einsatzgebieten von Stud.IP. Insgesamt gibt es derzeit ca. 70 Installationen mit ungefähr 600.000 Nutzerinnen und Nutzern [20].

Stud.IP versucht durch stetige Verbesserungen und Anpassungen den Anforderungen der sich ändernden Gegebenheiten der digitalen Lehre gerecht zu werden. Dabei ist ein kontinuierlicher Dialog zwischen Lehrenden, Lernenden sowie Entwicklern essentiell.

Stud.IP erfüllt alle im vorherigen Abschnitt 2.1 aufgeführten Anforderungen an ein LMS. Neben einer ausgereiften Benutzer- und Kursverwaltung inklusive eines Rollen- und Rechtessystems, existiert ebenfalls ein Forum und ein Chat zur Kommunikation sowie diverse Lernwerkzeuge (Etherpad, Wiki, Courseware, Vips). Stud.IP verfügt über diverse Werkzeuge sowie Schnittstellen zur Integration von externen Systemen und Anwendungen. Im Folgenden werden die für diese Arbeit relevanten Schnittstellen kurz beschrieben.

### 2.2.1 Stud.IP Plugins

Die Lernplattform Stud.IP besitzt eine Plugin-Schnittstelle, mit dessen Hilfe das LMS um beliebige Funktionen und Werkzeuge erweitert werden kann [15]. Der Grund hierfür sind die mitunter sehr individuellen Anforderungen jedes Standorts, an dem Stud.IP betrieben wird. Mit Hilfe von Plugins lässt sich die Software bedarfsgenau konfigurieren ohne direkte Änderungen am Kern-Programmcodex vornehmen zu müssen.

Plugins werden innerhalb von Stud.IP über den sogenannten Pluginmarktplatz bereitgestellt und lassen sich als Administrator einfach installieren und aktivieren bzw. deinstallieren oder deaktivieren.

Ein Ausschnitt der Administrationsoberfläche für die Pluginverwaltung ist in Abbildung 2.2 zu





Verwaltung von Plugins						28 Plugins (28/0)
Aktiv	Name	Typ	Version	Schema	Position	Aktionen
<input checked="" type="checkbox"/>	ActivityFeed <i>(Kern-Plugin)</i>	PortalPlugin	1.0		8	
<input checked="" type="checkbox"/>	Blubber <i>(Kern-Plugin)</i>	CorePlugin, StandardPlugin, StudipModule	3		1	
<input checked="" type="checkbox"/>	ContentsWidget <i>(Kern-Plugin)</i>	PortalPlugin	1.0		9	
<input checked="" type="checkbox"/>	CoreAdmin <i>(Kern-Plugin)</i>	CorePlugin, StudipModule			1	

Abbildung 2.2: Stud.IP Pluginverwaltung

sehen. Zusätzlich zu dem Namen, der Version, dem Status und dem Aktionsmenü wird ebenfalls der Plugintyp angezeigt, von dem es verschiedene gibt, worauf ich im Implementationsteil weiter eingehen werde. Hier sei nur kurz die Besonderheit von Kern-Plugins erwähnt, die bereits bei der Standardinstallation von Stud.IP mitgeliefert und installiert werden.

### 2.2.2 JSON:API

Die JSON:API ist eine Spezifikation, die Konventionen dafür festlegt, wie ein Client den Abruf oder die Änderung von Ressourcen anfordert (Request) und wie ein Server auf diese Anforderungen reagieren soll (Response). Die allgemeine JSON:API-Spezifikation wurde entwickelt, um sowohl die Anzahl der Anfragen als auch die Menge der zwischen Clients und Servern übertragenen Daten zu minimieren. Dieses Effizienzziel wird erreicht, ohne die Lesbarkeit, Flexibilität oder Auffindbarkeit zu beeinträchtigen [10].

Stud.IP verfügt über eine solche JSON:API-Schnittstelle, welche die oben genannte allgemeine Spezifikation erfüllt und mit deren Hilfe diverse Daten abgerufen sowie dem System hinzugefügt können.

Für Stud.IP existieren verschiedenste JSON:API-Routen, die unter folgender URI erreichbar sind:

```
https://<meine.studip.installation.de>/jsonapi.php/v1/<routen>
```

Ein Beispielaufruf zur Anforderung der Daten eines bestimmten Semester (eindeutigen ID) hat diese Form:

```
https://<meine.studip.installation.de>/jsonapi.php/v1/semester/<ID>
```

Die zugehörige Antwort im JSON-Format sieht wie folgt aus:

```

1  {
2    "data": {
3      "type": "semesters",
4      "id": "322f640f3f4643ebe514df65f1163eb1",
5      "attributes": {
6        "title": "SS 2021",
7        "description": "",
8        "token": "",
9        "start": "2021-04-01T00:00:00+02:00",
10       "end": "2021-09-30T23:59:59+02:00",
11       "start-of-lectures": "2021-04-12T00:00:00+02:00",
12       "end-of-lectures": "2021-07-16T23:59:59+02:00",
13       "visible": true
14     },
15     "links": {
16       "self": "\trunk\jsonapi.php\v1\semesters\322f640f3f4643ebe514df65f1163eb1"
17     }
18   }
19 }
```

Der Client bekommt die JSON-Repräsentation eines Stud.IP-Semesters zurückgeliefert, welche alle relevanten Daten der Objekts enthält. Routen repräsentieren die verschiedenen Stud.IP-

Datenstrukturen und werden mit Hilfe einer `RouteMap` von der angefragten URI auf den entsprechenden Programmcode abgebildet. Die Abbildung 2.3 stellt schemenhaft den Ablauf der Abarbeitung eines JSON:API-Requests dar. Nach einem initialen Request folgt die bereits an-

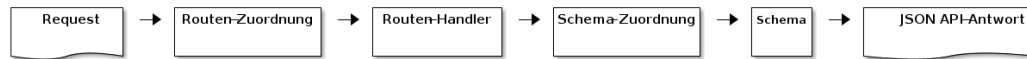


Abbildung 2.3: Stud.IP JSON:API Ablaufdiagramm

gesprochene Routen-Zuordnung, welche den Programmcode bestimmt, der ausgeführt werden soll. Der Routen-Handler kümmert sich um eine JSON:API-konforme Antwort. Die folgende Schema-Zuordnung legt fest welche Schemaklasse bestimmte Stud.IP-Objekte in JSON umwandeln kann. Im nächsten Schritt wird eben dieses Schema ausgewählt, welches die konkrete Abbildung eines Stud.IP-Objekts in JSON definiert. Daraufhin kann die JSON:API-Antwort an den Client gesendet werden [16].

### 2.2.3 Blubber-Chat

Innerhalb von Stud.IP gibt es neben dem Forum und den internen Nachrichten ebenfalls das Kommunikationswerkzeug Blubber, welches es ermöglicht mit anderen Personen Textnachrichten auszutauschen.

Blubber kann als eine Art integrierter Instant Messenger beschrieben werden, da Nachrichten sofort sichtbar sind und sich so präsentieren, wie man es von anderen Chat-Clients kennt. Allerdings gibt es derzeit keinen alleinstehenden Blubber-Client.

Es kann zudem auf Nachrichten anderer Nutzer geantwortet werden, was in einem voranstehenden Zitat der Ursprungsnachricht in der eigenen Nachricht führt. Außerdem können Nachrichten editiert und gelöscht werden.

In Abbildung 2.4 ist der globale Blubber einer Testinstallation inklusive einigen Beispielnachrichten dargestellt, der die zuvor genannten Eigenschaften und Funktionen aufzeigt. Unter dem Punkt **Konversationen** in der Sidebar werden alle eigenen Blubber aufgelistet und es können weitere hinzugefügt werden.

Dabei sind grundsätzlich drei verschiedene Arten von Blubber-Chats zu unterscheiden: öffentliche Blubber, private Blubber und Veranstaltungsblubber.

Die jeweilige Bezeichnung macht deutlich, wer alles Zugriff auf das Blubber hat und Nachrichten lesen kann. Nachrichten in einem öffentlichen Blubber kann jeder lesen und schreiben, der dem Blubber beitrifft. Direktnachrichten entsprechen den privaten Blubbern, zu denen gezielt Personen hinzugefügt werden können. Schreibt man eine Nachricht in einem Veranstaltungsblubber, so ist diese für alle Teilnehmenden der jeweiligen Veranstaltung sichtbar.[18]

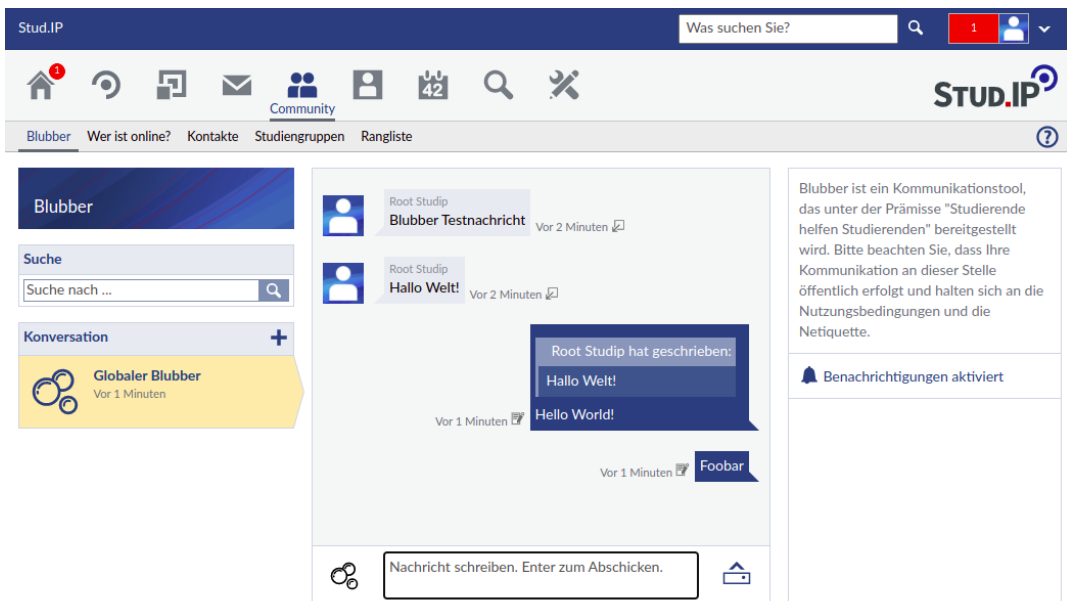


Abbildung 2.4: Stud.IP: Globaler Blubber

## 2.2.4 Notifications

Notifications sind ein Eventsystem für Stud.IP. Mit Hilfe der `NotificationCenter`-Klasse können an den entsprechenden Stellen im Programmcode Notification-Events wie folgt ausgelöst werden:

```
NotificationCenter::postNotification("user_accepted_to_seminar", $username);
```

Will man nun auf dieses Event reagieren (zum Beispiel innerhalb eines Plugins), so kann dies mit der `NotificationCenter::addObserver()`-Funktion erfolgen. Im folgenden Codebeispiel wird dies noch einmal genauer veranschaulicht.

```
1  <?php
2  class MyPlugin extends StudIPPlugin implements SystemPlugin {
3      public function __construct() {
4          NotificationCenter::addObserver($this,
5              "send_mail_to_accepted_user",
6              "user_accepted_to_seminar");
7      }
8
9      public function send_mail_to_accepted_user($username) {
10         /* hier das, was getan werden soll, wenn der Nutzer registriert ist */
11     }
12 }
```

Durch das Hinzufügen eines Observers kann man auf jedes beliebige Event, dass eine Notification



absetzt reagieren.[17]

## 2.3 Anbindung von Fremdsystemen an Stud.IP

Im Kontext des Lernmanagementsystems Stud.IP existieren mehrere Beispiele für die Anbindung oder Einbettung von anderen Systemen an bzw. in die LMS-Software. Im Folgenden werden ein paar solcher Integrationen konkret aufgeführt.

### 2.3.1 Meetings Plugin

Meetings ist ein Stud.IP Videokonferenzplugin [6], das das LMS mit unterschiedlichen Videokonferenzsoftwarelösungen, wie z.B. BigBlueButton [2], verknüpfen kann. Das Plugin ermöglicht es direkt aus einer Stud.IP-Veranstaltung heraus Videokonferenzräume für z.B. Vorlesungen, Webinare, Vorträge oder Gruppenarbeiten anzulegen, denen die Teilnehmenden der Veranstaltung beitreten können (dank der Rechteverwaltung in Stud.IP). Per Einladungslink ist es auch Personen ohne ein Stud.IP-Nutzerkonto (z.B. Gastredner) möglich an Videokonferenzen teilzunehmen. Neben den zu erwartenden Kommunikationswerkzeugen wie Mikrofon, Kamera und einer Chatfunktion können zusätzlich gemeinsam Präsentationen angesehen, Texte im integrierten Etherpad kooperativ erstellt und der eigene Bildschirm für alle Teilnehmenden freigeschaltet werden. Das Plugin ist somit eine ideale Erweiterung für dezentrales Lernen und Arbeiten, insbesondere da auch aus einfachen Stud.IP-Studiengruppen heraus Meetings-Räume erstellt werden können, außerhalb des strikten Lehrveranstaltungskontextes. Dies hilft Lehrenden bei der Vernetzung und Studierenden bei der Bildung von Arbeits- und Lerngruppen.

Weiterhin besteht auch die Möglichkeit Videokonferenzen aufzuzeichnen, was die Nacharbeitung oder Wiederholung von Lehrinhalten bei einem verpassten Meeting um ein Vielfaches erleichtert. [19]

Die folgende Abbildung 2.5 zeigt einen Screenshot des „Raum erstellen“-Dialogs in dem Meetings Plugin mit all seinen Optionen.

Neben den Grundeinstellungen wie **Raumname** und **Systemeinstellung** können noch diverse andere Konfigurationsoptionen ausgewählt werden, die je nach selektiertem Konferenzsystem unterschiedlich ausfallen. Die Abbildung 2.5 verdeutlicht welche Auswahlmöglichkeiten in BigBlueButton zur Verfügung stehen und stellt die Benutzeroberfläche dar, die als Grundlage der im Folgenden beispielhaft beschriebenen BigBlueButton API dient.

Um einen API-Aufruf an den BigBlueButton-Server zu senden, stellt man eine HTTPS-Anfrage an den API-Endpunkt des BigBlueButton-Servers (normalerweise bestehend aus dem Hostnamen des Servers gefolgt von `/bigbluebutton/api`). Alle API-Aufrufe müssen eine Prüfsumme enthalten, die mit Hilfe eines mit dem BigBlueButton-Server geteilten Geheimnis berechnet wird.

Der BigBlueButton-Server gibt daraufhin eine Antwort im XML-Format für alle API-Aufrufe zurück. Beispielhaft ist hier der API-Aufruf zum Erstellen eines Meetings in BigBlueButton

**Raumkonfiguration** ✕

**Raumname**

**Konferenz Systemeinstellung**

**Konferenzsystem \***

BigBlueButton ▼

**Verfügbare Server \***

Bitte wählen Sie einen Server aus ▼

**Zusätzliche Funktionen**

- ☐ Alle Teilnehmenden haben Moderationsrechte
- ☒ Alle Teilnehmenden initial stumm schalten
- ☒ Jeder Teilnehmer kann die Konferenz starten
- ☐ Nur Moderatoren können Webcams sehen ⓘ
- ☐ Nur Moderatoren können Webcams teilen
- ☐ Nur Moderatoren können Audio teilen
- ☐ Gemeinsame Notizen deaktivieren
- ☐ Private Chats deaktivieren
- ☐ Moderatoren vor Teilnehmendenzutritt fragen ⓘ
- ☐ Zugang via Link ⓘ

Maximale Teilnehmerzahl

Minuten Konferenzdauer (Max. Limit: 1440 Minuten) ⓘ

Willkommensnachricht ⓘ

Welcome to <b>%%CONFNAME%%</b>!  
<br><br>For help on using B

**Aufzeichnung**

- ☐ Sitzungen können aufgezeichnet werden. **beta**
- ☒ Aufzeichnungen für Teilnehmende sichtbar schalten ⓘ

**Automatisches hochladen von Materialien** ⓘ

Aktuell ausgewählter Ordner: Kein Ordner

**Name**

- Allgemeiner Dateiordner
- Aufgaben-Plugin
- Martins
- nils
- Ordner tests

**Abbildung 2.5:** Meetings-Plugin: Dialog bei Raumerstellung

aufgeführt (create):

`http://mein.server.de/bigbluebutton/api/create?[parameters]&checksum=[checksum]` [1]

Die konkrete Implementation auf Programmcodenebene sieht folgendermaßen aus:

```

1  <?php
2  public function createMeeting(MeetingParameters $parameters)
3  {
4      $params = array(
5          'name' => $parameters->getMeetingName(),
6          'meetingID' => $parameters->getRemoteId() ?: $parameters->getMeetingId(),
7  
```

```

8      // ...
9      );
10
11     // ...
12     $response = $this->performRequest('create', $params, $options);
13
14     $xml = new \SimpleXMLElement($response);
15     // ...
16     return isset($xml->returncode) && strtolower((string)$xml->returncode) === 'success';
17 }
18
19 private function performRequest($endpoint, array $params = array(), array $options = [])
20 {
21     $uri = 'api/'. $endpoint. '?' . $this->buildQueryString($params);
22
23     //...
24
25     $method = (is_array($options) && count($options)) ? 'POST' : 'GET';
26     $request = $this->client->request($method, $this->url . '/' . $uri, $options);
27     return $request->getBody(true);
28 }

```

Zum besseren Verständnis werden hier nur die relevanten Abschnitte und Methoden gezeigt. Die Funktion `createMeeting()` sammelt zunächst alle Parameter, die zur Erstellung eines Meetings relevant sind und deren Werte zuvor in der Nutzeroberfläche ausgewählt wurden.

In Zeile 12 wird anschließend die Methode `performRequest()` mit dem entsprechenden Schlüsselwort `create` und den Parametern aufgerufen. Bei einem Erfolg wird `true` zurückgegeben.

Die Funktion `performRequest()` wird ab Zeile 19 beschrieben und zeigt auf, wie die konkrete Serveranfrage mit dem jeweiligen Endpoint entsprechend der oben skizzierten allgemeinen Spezifikation zusammengestellt wird. Am Ende wird der die XML-Antwort des Servers im Response-Body zurückgegeben.

Damit ist die Funktionalität des Stud.IP Meetings Plugins grundsätzlich gezeigt und die Anbindung von BigBlueButton mit Hilfe der API deutlich gemacht.

### 2.3.2 Etherpad Plugin

Die webbasierte Open-Source-Software Etherpad ist ein kollaborativer Echtzeit-Editor, der das Bearbeiten eines Dokuments durch viele Nutzer gleichzeitig erlaubt [5].

Jedem Teilnehmenden wird dazu eine eindeutige Farbe beim Editieren des Textes zugewiesen. Die integrierte Versionierung macht es zudem einfach vorherige Änderungen nachzuvollziehen oder auf eine alte Version zurückzugreifen, falls nötig. Damit eignet sich Etherpad insbesondere gut für Brainstorming, Protokolle, Sammlungen, Gruppen- und Seminararbeiten oder ähnliches.

Das Etherpad Plugin ist ein weiteres Modul mit dem das externe Programm Etherpad in Stud.IP eingebunden wird. Das Plugin verbindet die Lernplattform mit einem Etherpad-Lite Server, der in vielen Fällen von der jeweiligen Institution selbst gehostet werden kann.

Dozenten oder Tutoren beliebig viele der sogenannten Pads in einer Stud.IP-Veranstaltung bzw. -Studiengruppe anlegen. Innerhalb einer Veranstaltung können die Teilnehmenden anschließend die bereits existierenden Pads zum gemeinsamen Arbeiten nutzen.

Die technische Umsetzung der Einbettung des Editors in Stud.IP sieht folgendermaßen aus und geschieht mit Hilfe eines `iframe`, dessen Ausmaße auf sinnvolle Weise beschränkt werden.[7]

```

1 <iframe id="etherpad"
2     src="<?= $controller->link_for('pads/open', $padid) ?>"
3     style="width:100%">
4 </iframe>
5 <script>
6 jQuery(window).on('message onmessage', function (e) {
7     var msg = e.originalEvent.data;
8     if (msg.name === 'ep_resize') {
9         var width = msg.data.width;
10        var height = msg.data.height;
11        jQuery("#etherpad").height(Math.min(Math.max(height, 400), 1080))
12    }
13 });
14 </script>

```

Die Benutzeroberfläche eines konkreten Pads einer Veranstaltung in Stud.IP ist in Abbildung 2.6 dargestellt. Neben dem eingebetteten Editor mit seinen Formatierungsmöglichkeiten verfügt das

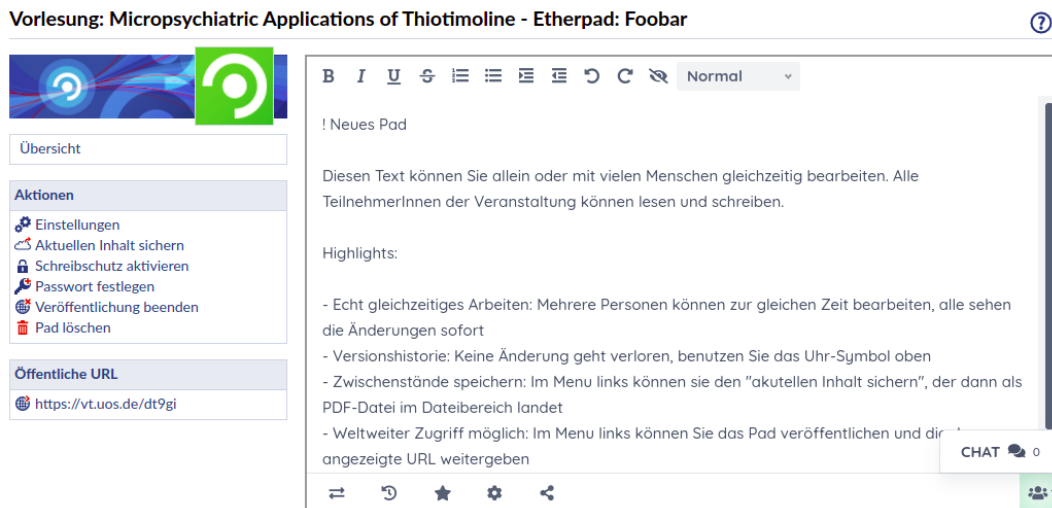


Abbildung 2.6: Etherpad: Einbettung mit `iframe`

Plugin über ein Stud.IP-Aktionsmenü mit erweiterten Funktionen wie beispielsweise dem PDF-

Export des Textes („Aktuellen Inhalt sichern“) und einer Veröffentlichungsfunktion, wodurch das Pad auch außerhalb von Stud.IP per generiertem Link zu erreichen ist.

### 2.3.3 Moodle-Connect Plugin

Mit Hilfe des Moodle-Connect Plugins lassen sich Stud.IP-Veranstaltungen mit Kursen des Lernmanagementsystems Moodle verknüpfen [8].

Es können bereits existierende Moodle-Kurse mit einer Veranstaltung in Stud.IP verbunden werden insofern man Dozentenrechte auf beiden Lernplattformen besitzt. Gibt es in Moodle noch keine passende Veranstaltung, so wird ein neuer Kurs automatisch angelegt. Dabei wird eine Veranstaltung aus Stud.IP in Moodle mitsamt seiner Teilnehmer übernommen.

Die Abbildung 2.7 zeigt die Benutzeroberfläche des Plugins aus Dozentensicht. Wie bereits be-

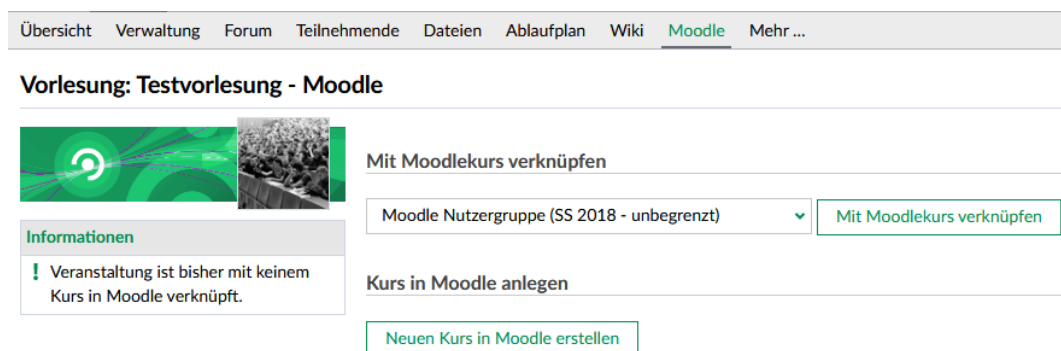


Abbildung 2.7: Dozentensicht: Moodle-Connect Plugin [21]

schrieben, sind hier die beiden Hauptfunktionen (Kurs verknüpfen/anlegen) zu sehen sowie die Möglichkeit die erstellte Verknüpfung wieder aufzuheben.

Im Folgenden soll grob auf die Funktionsweise des Plugins eingegangen werden. Zum reibungslosen Ablauf muss ein Administrator einige Einstellungen in Moodle sowie dem Stud.IP Plugin vornehmen. Nach der erfolgreichen Aktivierung der REST-API-Schnittstelle und der Erstellung eines API-Tokens in Moodle, werden diese Werte (MOODLE\_API\_URI und MOODLE\_API\_TOKEN) in der Plugin Konfiguration von Stud.IP eingetragen. Anschließend kann die Moodle-API aus Stud.IP heraus aufgerufen und genutzt werden. Das folgende Codebeispiel zeigt die Erstellung einer neuen Veranstaltung in Moodle, wenn in der Benutzeroberfläche „Neuen Kurs in Moodle erstellen“ ausgewählt wurde.

```

1  <?php
2  function create_action()
3  {
4      // ...
5      $data = ['courses' => [

```

```

6      [
7          'fullname' => studip_utf8encode(
8              $this->course->getFullname('number-name-semester')),
9          'shortname' => md5(uniqid())
10         // ...
11     ]
12 ];
13
14 $response      = Moodle\REST::post('core_course_create_courses', $data);
15 $moodle_course = array_pop($response);
16
17 // ...
18 }

```

Zunächst werden die für Moodle erforderlichen Kursdaten in einem Array zusammengetragen (Zeile 5-12), das anschließend per API-Request mit an Moodle übergeben wird (Zeile 14). Die anschließende Serverantwort liefert eine Erfolgs- oder Misserfolgsmeldung zur Weiterverarbeitung zurück.

## 2.4 Matrix-Messenger

Messenger sind in der Regel netzwerkbasierte (meist webbasierte) Programme zur Übermittlung von Textnachrichten in Echtzeit zwischen zwei oder mehr Teilnehmenden, auch Instant Messaging genannt.

Dabei stößt der Absender ein Push-Event an, das die versendete Nachricht möglichst unmittelbar beim Empfänger ankommen lässt. Zum tatsächlichen Verschicken und Erhalten von Nachrichten benötigen die Nutzer meist ein Client-Programm, das die Nachrichten zunächst an einen Server schickt, welcher sie zum jeweiligen Adressaten weiterleitet.

Ein konkretes Beispiel eines solchen Messenger-Server-Backends stellt Matrix dar. Matrix ist ein offener Standard für interoperable, dezentralisierte Echtzeitkommunikation über IP. Er kann für Instant Messaging, VoIP/WebRTC-Nachrichtenübermittlung, Kommunikation im Internet of Things oder überall dort eingesetzt werden, wo eine Standard-HTTP-API für die Veröffentlichung und das Abonnieren von Daten bei gleichzeitiger Verfolgung des Gesprächsverlaufs benötigt wird.

Matrix definiert den Standard und stellt Open-Source-Referenzimplementierungen von Matrix-kompatiblen Servern, Client-SDKs und Anwendungsdiensten zur Verfügung, die dabei unterstützen können, neue Kommunikationslösungen zu entwickeln oder die Fähigkeiten und Reichweite bestehender Lösungen zu erweitern.[13]

### 2.4.1 Matrix Schnittstellen

Matrix stellt verschiedene offene HTTP-API-Schnittstellen für die Übermittlung von JSON-Nachrichten zur Kommunikation nach außen bereit. Neben einer Server-Server-API-Schnittstelle, die definiert wie unterschiedliche Matrix Homeserver Nachrichten miteinander austauschen und synchronisieren wird ebenfalls eine Client-Server-API angeboten, die festlegt, wie Matrix-kompatible Clients mit einem Matrix Homeserver kommunizieren.[13]

Lediglich die zuletzt genannte API ist für diese Arbeit relevant. Nachfolgend wird kurz darauf eingegangen, wie die API grundsätzlich zu benutzen ist.

Zunächst wird ein Zugangstoken (`access_token`) benötigt um erfolgreich Anfragen an die Matrix-API stellen zu können. Dazu braucht man einen Account auf dem Homeserver der die Anfrage empfangen soll. Mit folgendem Aufruf erhält man ebendiesen Zugangstoken

```
curl -XPOST -d '{"type":"m.login.password", "user":"manschwarz", "password":<PASSWORD>}',
"https://matrix.org/_matrix/client/r0/login"
```

Die zurückgelieferte Antwort im JSON-Format sieht folgendermaßen aus:

```
1 {
2   "access_token": "QGV4YW1wbGU6bG9jYWxob3N0.vRDLTgxefmKWQEtgGd",
3   "home_server": "matrix.org",
4   "user_id": "@manschwarz:matrix.org"
5 }
```

Mit diesem `access_token` lassen sich nun diverse Requests an den Homeserver stellen. Die generelle API-Aufrufstruktur ist (wie im vorherigen Beispiel bereits angedeutet) wie folgt aufgebaut:

```
<HTTP-Method> https://<matrix.server>/_matrix/client/r0/<ROUTE> <JSON-Data>
```

Die Client-Server-API bietet eine einfache, leichtgewichtige API, mit der Clients Nachrichten senden, Räume steuern und den Gesprächsverlauf synchronisieren können.[12]

### 2.4.2 Matrix Client

Aufgrund der offenen API-Schnittstellen gibt es eine große und stetig wachsende Auswahl von Matrix-kompatiblen Clients. Ein sehr verbreiteter und auch an der Universität Osnabrück eingesetzter Webclient ist Element [4]. Neben diversen weiteren mobilen, Desktop-, Web- und sogar Terminal-basierten Chat-Clients existieren sogenannte Bridge-Lösungen für fast alle gängigen Messenger (z.B. WhatsApp, Telegram, Signal oder Discord) [11].





## Kapitel 3

# Anforderungsanalyse

Die Anforderungsanalyse und die daraus resultierende Erstellung eines Anforderungskatalogs zum Thema „Stud.IP-Matrix-Integration“ sowie die sinnvolle Umsetzung dessen bilden die Grundlage dieser Ausarbeitung.

Hierzu wurde im Rahmen dieser Arbeit eine kleine interne Umfrage unter den Mitarbeitern und Mitarbeiterinnen des Zentrums virtUOS durchgeführt.

### 3.1 Umfrage zur Anforderungsanalyse

Die Umfrage wurde mit dem webbasierten Tool Google-Forms erstellt, was die Konzeption, Distribution und Auswertung erheblich erleichtert sowie die Anonymität der Teilnehmenden wahrt. Die Abbildung 3.1 zeigt einen Screenshot der überschaubaren Forms-Erhebung mit ihren zwei Fragen.

Neben der im Mittelpunkt stehenden Freitextfrage zu den gewünschten Funktionen einer Anbindung von Matrix an Stud.IP dient die zweite Frage einer ungefähren Tätigkeitsbereichszuordnung der abstimmenden Personen.

Das Ergebnis letzterer Frage ist in Abbildung 3.2 zu sehen. Das Diagramm zeigt eine relativ ausgeglichene Verteilung und keine überproportionale Repräsentation eines bestimmten Tätigkeitsbereichs.

Insgesamt haben 20 Personen an der Umfrage teilgenommen. Aus den eingegangenen Antworten wurde ein Anforderungskatalog erstellt auf den im Kapitel 3.3 genauer eingegangen wird.

Alle Antworten im Detail befinden sich im Anhang A.

## Anforderungsanalyse

Im Zuge der Abschlussarbeit "Anbindung von Messaging-Systemen an Lernmanagementsysteme (am Beispiel von Stud.IP und Matrix)" soll diese kleine empirische Erhebung dabei helfen einen Anforderungskatalog für die Umsetzung zu erstellen.

Was wäre Ihnen bei der Anbindung von Matrix (Element) an Stud.IP (Blubber) wichtig? Welche Anforderungen oder Funktionen sollten Ihrer Meinung nach erfüllt werden? \*

Long answer text

Was trifft am ehesten auf Sie zu? \*

☐ Ich bin in der Softwareentwicklung tätig.

☐ Ich bin in der Konzeption und/oder Beratung tätig.

☐ Ich bin Anwender\*in von Stud.IP und/oder Matrix/Element.

Abbildung 3.1: Umfrage: Anforderungsanalyse

Was trifft am ehesten auf Sie zu?

20 responses

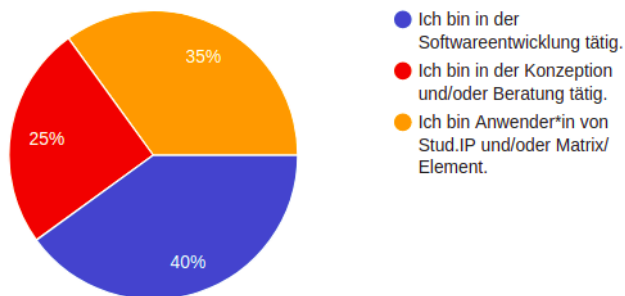


Abbildung 3.2: Umfrage: Teilnehmendenverteilung

## 3.2 Grundsatzentscheidung

Bevor der Anforderungskatalog im einzelnen abgearbeitet wird, soll an dieser Stelle kurz die Grundsatzentscheidung diskutiert werden, die der weiteren Bearbeitung zugrunde liegt.

Bei dem Entwurf einer Anbindung von Matrix an Stud.IP sind prinzipiell zwei Konzepte entstanden, die beschreiben, wie eine konkrete Umsetzung aussehen könnte.

Das erste Konzept stellt eine Implementation mit möglichst vollständigem Funktionsumfang eines eigenen Matrix-Clients in Stud.IP dar. Dazu könnte Blubber genutzt und erweitert oder umgebaut werden.

Schaut man sich einmal den Element Webclient und das entsprechende Github-Repository [4] als eine Referenz für einen solchen Client an, so lässt sich schnell erkennen, dass dies ein großes Softwareprojekt mit vielen Mitwirkenden ist. Zusätzlich zu den über 14.000 LOC (Lines of Code) sprechen die 5.000 offenen Issues ebenfalls für eine hohe Komplexität und einen hohen Pflegeaufwand des Projekts.

Eine solche Umsetzung ist im Hinblick auf eine ehrliche Aufwandsabschätzung im Zuge dieser Arbeit nicht realistisch. Selbst wenn ein eigener Client entwickelt würde, bestünde noch die Frage, wer sich um die weitere Pflege und notwendige Updates kümmert. Zudem ist mit Element ein sehr guter Client vorhanden, was es teilweise überflüssig macht alles erneut zu implementieren.

Das zweite Konzept beschreibt eine lose Kopplung von Matrix an Stud.IP: eine Anbindung anstatt einer Integration.

Konkret könnte dies so aussehen, dass weiterhin zwei klar getrennte Systeme existieren, was der Transparenz zuträglich ist, und lediglich Chat-Räume in Matrix aus einer Stud.IP-Veranstaltung heraus erstellt und benutzt werden können (ähnlich wie beim Moodle-Connect Plugin aus Kapitel 2.3.3).

Weiterhin unterstützt dieser Ansatz auch die Loslösung von Blubber und die Migration zu Element, da der Stud.IP-interne Nachrichtendienst nicht viel genutzt wird (zumindest an der Universität Osnabrück).

Aus den oben genannten Gründen habe ich mich bei der weiteren Bearbeitung des Themas für das zweite Konzept entschieden.

## 3.3 Anforderungen

Dieses Kapitel beschäftigt sich mit den aus der Umfrage (Kapitel 3.1) herausgearbeiteten Anforderungen. Diese bilden die Grundlage der Implementation im nachfolgenden Kapitel 4.

In den Antworten wurden einige Anforderungen genannt, die im Folgenden nicht explizit behandelt werden, da diese bereits im Element-Client vorhanden sind. Insbesondere eine übersichtliche und moderne Benutzeroberfläche, Nachrichten senden und empfangen, Portabilität (Unabhängigkeit von Betriebssystem oder Browser) und eine klare Bedienung (wenige Klicks zum Bilden von Arbeitsgruppen und Direktchats). Zudem soll Matrix als Hauptnachrichtendienst fungieren, was ebenfalls mit dem zweiten Konzept einhergeht, obwohl die Dozenten letztendlich entscheiden

was genutzt wird. Im Extremfall könnten die Betreiber Blubber sogar abschalten, wenn Matrix von den Nutzern gut angenommen wird.

Das Stummschalten von Chat-Räumen und als gelesen markierte Nachrichten sind in Element ebenfalls integriert und in Stud.IP bisher nicht vorgesehen.

Der sich einmalig gewünschte Support-Hotline-Raum kann ebenfalls schon in Element realisiert werden, indem man einfach einen für alle offenen Raum anlegt, in dem die Support-Mitarbeiter ebenfalls drin sind und Fragen beantworten.

Die folgenden Anforderungen sind auf einzelne Kapitel aufgeteilt und werden separat diskutiert.

### 3.3.1 Veranstaltungen auf Räume abbilden

Kurse und Studengruppen sollten (automatisch) auf Räume in Matrix abgebildet werden. Nach der Aktivierung des Matrix-Plugins ist der Reiter für den Matrix-Chat in einer Veranstaltung zunächst nur für Dozenten und Tutoren sichtbar. Nachdem diese die Generierung eines Matrix-Raums angestoßen haben wird der Reiter auch für die normalen Teilnehmenden sichtbar. Die Studierenden finden auf dem Reiter unter anderem eine kurze Anleitung und Hinweise, zu den übermittelten Daten zu Matrix sowie einen Button für die Teilnahme am Matrix-Raum.

### 3.3.2 Übernahme von Rollen, Rechten und Status

Aus Datenschutzgründen und weil die Sichtbarkeitseinstellungen aus Stud.IP in Matrix nicht abbildbar sind (in Matrix kann ein Nutzer z.B. nicht unsichtbar sein), sollte keine automatische Übernahme der Teilnehmerliste in einen neu angelegten Matrix-Raum stattfinden. Stattdessen müssen die Studierenden wie im vorherigen Kapitel beschrieben den Beitritt in einen Matrix-Raum selbst initiieren. Dadurch erübrigt sich auch die Frage, was mit später hinzugekommenen und ausgetretenen Personen passieren soll, bzw. spart man sich aufwändige Synchronisationsprozesse der Teilnehmerlisten zwischen Stud.IP und Matrix.

Weiterhin sind Teilnehmer einer Veranstaltung in der Teilnehmerliste in Matrix immer sichtbar und man sieht neben dem aktuellen Status (online, offline seit 2 Std., usw.) auch, wer wann welche Nachricht gelesen hat. Da laut Konzept lediglich eine lose Kopplung vorherrscht, wird der Matrix-Raum einmal angelegt und ist danach von Stud.IP unabhängig.

Feingranulare Berechtigungen für Studierende ist nur eingeschränkt umsetzbar, da es in Matrix nur drei Rechtstufen gibt. Eine denkbare Rechteverteilung wäre Dozenten auf **Admin**, Tutoren auf **Moderator** und Studierende auf **Default** abzubilden.

In Matrix lassen sich zwar zusätzlich benutzerdefinierte Rechtstufen erstellen, aber dies wurde im Zuge dieser Arbeit nicht weiter betrachtet.

### 3.3.3 kein zusätzliches Login

Es sollte lediglich ein Login für beide Systeme (Stud.IP/Matrix) geben. Single Sign-On Stud.IP Nutzer sollten ohne zusätzliches Login in einen Matrix Raum gelangen können. Dies ist dank der LDAP-Authentifizierung in Stud.IP und einer möglichen LDAP-Anbindung an Matrix gut umzusetzen und wird so auch bereits an der Universität Osnabrück betrieben.

### 3.3.4 Benachrichtigungsfunktion

Man möchte in Matrix über relevante Stud.IP-Ereignisse informiert werden. Hilfreich wäre hier eine Relevanzabschätzung für verschiedene Ereignisse zu erstellen um unter Umständen festzustellen was relevant für alle Teilnehmenden eines Raums ist, und was nicht. Persönliche Präferenzen kann man mit dem hier entschiedenen Modell nicht berücksichtigen.

Denkbar wäre eine Konfiguration pro Veranstaltung, was im Matrix-Raum erscheinen soll und was nicht. Dabei stellt sich die Frage nach der Sinnhaftigkeit insbesondere aus Studierenden-sicht, wenn in unterschiedlichen Veranstaltungen ein Ereignis einmal angezeigt wird und einmal nicht. Dies führt vermutlich eher zu Verwirrungen, weshalb es keine veranstaltungsbezogene Konfiguration geben wird.

Die Richtung der Benachrichtigungen geht ausschließlich von Stud.IP zu Matrix und nicht umgekehrt. Grund hierfür ist die bereits genannte lose Kopplung und sei hier nur der Vollständigkeit halber genannt. Zudem ist eine globale Informationsveranstaltung mit universitätsweiten Infos denkbar. Allerdings ist die Benachrichtigungsfunktion für veranstaltungsübergreifende Informationen, wie z.B. Rückmeldefristen, Unischließungen und ähnliches eher ungeeignet, da tatsächlich so gut wie alle Studierenden in diesem Infochannel sein müssten in dem keiner schreiben darf. Es wäre lediglich eine Einwegkommunikation und dafür ist vermutlich das interne Nachrichtensystem in Stud.IP besser geeignet. Veranstaltungsbezogene Informationen wie z.B. das hochladen einer Datei oder der Ausfall eines Termins sind für eine solche Benachrichtigungsfunktion deutlich besser geeignet.

Synchronizität zwischen Matrix und Blubber ist aufgrund der losen Kopplung nicht gegeben.

### 3.3.5 Transparenz

Es sollte die Verknüpfung von Stud.IP und Matrix kenntlich gemacht werden. Die Konsequenzen in Matrix, für das eigene Handeln in Stud.IP, sollten ersichtlich sein. Vorstellbar wäre ein Hinweis bei Aktionen, die Auswirkungen auf Matrix haben (z.B. "Diese Aktion wird in Matrix erscheinen."). Studierende sollten informiert werden, dass es einen Matrix-Raum gibt, beispielhaft durch einen kleinen Infotext auf dem Matrix-Chat-Reiter.

### 3.3.6 Direktnachrichten

Mit Direktnachrichten sind Eins-zu-eins Chats gemeint. Diese sind bereits in Matrix vorhanden. Man könnte die Stud.IP-Blubber Direktchats mit Matrix verbinden.

Ein Anwendungsszenario wäre z.B. eine Sprechstundenfunktion, bei der man für eine kurze Zeit einen 1-zu-1 Chat mit einer Person führt, die den Chat danach automatisch verlässt. Andere Personen sind in dieser Zeit in einer Warteschlange.

### 3.3.7 mehrere Räume pro Kurs

Es sollten aus einer Stud.IP Veranstaltung mehrere Matrix-Räume erstellt werden können, die dann als Community zusammengefasst werden könnten.

### 3.3.8 Kompatibilität von Nachrichten in Blubber und Matrix

Stud.IP (Blubber) sollte als regulärer Client agieren. Dieser Punkt ist durch die Umsetzung des zweiten Konzeptes bereits nicht erfüllbar.

#### **Dateien versenden**

Wenn in Matrix Dateien versendet werden, sollten diese in Stud.IP ebenfalls im Dateibereich landen und als Inhalt verlinkt werden. Aufgrund der losen Kopplung und keiner vollständigen Client Implementation wird auch auf diesen Punkt in der Umsetzung verzichtet.

#### **Emoticons korrekt anzeigen**

Da Element ein vollwertiger Chat-Client ist, stellt das Anzeigen von Emoticons kein Problem dar.

#### **Nachrichten sollten editier- und löschar sein**

Auch das ist in Matrix problemlos möglich.

### 3.3.9 Übernahme des Stud.IP Profilbildes

Bei einem Profilbild handelt es sich um personenbezogene Daten, was natürlich unter den Datenschutz fällt. Die Option zur Profilbildübertragung bedarf einer aktiven Zustimmung und kann ebenfalls auf dem Veranstaltungsreiter erfolgen. Dabei sind die Einstellungen unter Profil->Einstellungen->Privatsphäre->Eigenes Bild zu beachten.

### 3.3.10 Stautsindikator in Stud.IP

Es sollte ein Icon für den Reiter in Stud.IP geben, welcher kenntlich macht, wie viele Nachrichten es in Matrix seit dem letzten Besuch in Stud.IP gab.

### 3.3.11 Anbindung von Matrix durch ein Stud.IP-Plugin

Eine der nicht funktionalen Anforderungen stellt die Anbindung des Matrix-Chats an Stud.IP in Form eines Plugins dar. So kann jeder Standort für sich entscheiden, ob er diese Anbindung installieren und anbieten möchte.

### 3.3.12 Robustheit, Ausfallsicherheit

Man kann sich die Frage stellen, was passiert, wenn der Matrix-Server mal für gewisse Zeit ausfällt. Messages könnten in einer Datenbanktabelle gepuffert werden welche mit Hilfe eines Cronjobs einmal stündlich geprüft werden, ob sie schon abgearbeitet und verschickt wurden, oder nicht.





## Kapitel 4

# Implementation

Das folgende Kapitel beschreibt in Kürze die zur Implementation der jeweiligen Anforderung nötigen Schritte und diskutiert, falls nötig, die Sinnhaftigkeit einer Umsetzung und begründet auch, wenn eine Anforderung nicht umgesetzt wurde.

### 4.1 Veranstaltungen auf Räume abbilden

Dozenten und Tutoren können aus einer Stud.IP Veranstaltung heraus einen Matrix-Raum erzeugen. Dazu wird der folgende API-Call verwendet.

```
1 curl -XPOST -d '{"room_alias_name":"manschwa-tutorial"}'  
2 "https://matrix.org/_matrix/client/r0/createRoom?access_token=<TOKEN>"
```

Bei initialer Aktivierung des Plugins ist der Matrix-Reiter in der Veranstaltung nur für Dozenten und Tutoren sichtbar. Nachdem ein Raum erstellt wurde, können auch die Studierenden den Reiter sehen und über diesen der Veranstaltung beitreten.

### 4.2 Übernahme von Rollen, Rechten und Status

Da es in Matrix lediglich drei Rechtstufen gibt und kein so feingranulares System wie in Stud.IP, werden die Rollen zunächst wie folgt verteilt. Dozenten bekommen Admin-Rechte in den Matrix-Räumen, Tutoren werden zu Moderatoren und Studenten bekommen die Default-Rolle zugewiesen.

### 4.3 kein zusätzliches Login

Da Stud.IP bereits über ein Login über LDAP verfügt und man sich in Matrix mit Hilfe des Synapse LDAP Auth Providers [3] ebenfalls über LDAP einloggen kann, ist diese Anforderung bereits erfüllt und wird an der Universität Osnabrück auch bereits eingesetzt.

Dazu wird im Ansible Skript zum Aufsetzen eines Matrix-Synapse-Servers folgender Code benötigt.

```

1 password_providers:
2   - module: "ldap_auth_provider.LdapAuthProvider"
3     config:
4       enabled: true
5       uri: "ldap://ldap.uni-osnabrueck.de:389"
6       start_tls: true
7       base: "ou=people,dc=uni-osnabrueck,dc=de"
8       attributes:
9         uid: "uid"
10        mail: "mail"
11        name: "cn"

```

### 4.4 Benachrichtigungsfunktion

Wie in Kapitel 2.2.4 beschrieben kann zur Umsetzung eines Benachrichtigungssystems das `NotificationCenter` herangezogen werden. Beispielhaft sei hier der Code zur Registration an einem Event, das einem anzeigt, wenn in der Veranstaltung eine neue Datei hochgeladen wurde.

```

1 <?php
2 NotificationCenter::addObserver($this,
3   'notify_matrix_room',
4   'FileRefDidCreate');

```

Diese Notification wird dann zusammen mit dem Dateinamen in den Matrix-Raum geschrieben.

### 4.5 Transparenz

Eine gewisse Transparenz wird dadurch erreicht, dass die Nutzer vor dem Betreten eines Matrix-Raumes aus Stud.IP heraus darüber in Kenntnis gesetzt werden, welche Daten übertragen werden und was in Matrix sichtbar ist. Die Verknüpfung zwischen Stud.IP und Matrix/Element wird dadurch deutlich gemacht. Weiterhin könnte man Dialoge oder kurze Hinweistexte einblenden, wenn eine Aktion in Stud.IP zu Matrix propagiert wird. Dies wurde hier allerdings nicht umgesetzt. Man würde es ebenfalls mit dem `NotificationCenter` und dem Stud.IP-internen Eventsystem realisieren.

## 4.6 Direktnachrichten

Direktnachrichten sind bereits in Matrix vorhanden, weshalb hier kein Implementationsbedarf bestand.

## 4.7 mehrere Räume pro Kurs

Dieses Feature wurde ebenfalls nicht umgesetzt aber eine Mögliche Variante könnte so aussehen, wie in Blubber, siehe Kapitel 2.2.3. Man könnte mehrere Räume über Stud.IP in Matrix anlegen und diese dann in einer Liste in der Sidebar verwalten. In Matrix bestünde zusätzlich die Möglichkeit alle Räume einer Veranstaltung zu einem Space zusammenzufassen.

## 4.8 Kompatibilität von Nachrichten in Blubber und Matrix

Aufgrund der Konzeptentscheidung und des bereits vorhandenen, großen Funktionsumfangs von Matrix und Element gibt es für diese Anforderung ebenfalls keinen Implementationsbedarf.

## 4.9 Übernahme des Stud.IP Profilbildes

Nachdem die Rechte des Profilbildes in Stud.IP korrekt eingestellt sind, kann das Profilbild mit Hilfe des folgenden API-Aufrufs zu Matrix übertragen werden.

```
PUT /_matrix/client/r0/profile/{userId}/avatar_url
```

## 4.10 Stautsindikator in Stud.IP

Aufgrund von Zeitmangel wurde diese Anforderung nicht implementiert. Es würde aber vermutlich auf eine Kombination der `objectsUserVisit`-Tabelle in Stud.IP und einem regelmäßig durchlaufendem Cronjob hinauslaufen.

## 4.11 Anbindung von Matrix durch ein Stud.IP-Plugin

Wie bereits im Anforderungskapitel 3.3.11 angesprochen soll die Anbindung von Matrix an Stud.IP mit Hilfe eines Plugins erfolgen, was die Option des direkten Einbaus in den Kern-Programmcode von vorn herein ausschließt.

Für die Implementation sind somit grundsätzlich zwei Varianten denkbar. Die erste Variante wäre die Umsetzung in einem Plugin und die zweite Variante wären mehrere Plugins, die

anschließend interagieren. Wie bereits in Abbildung 2.2 zu sehen war, existieren in Stud.IP verschiedene Typen von Plugins, wobei ein konkretes Plugin mehrere Typen haben kann. Der Typ **StandardPlugin** ermöglicht es zum Beispiel eigene Navigationspunkte in Veranstaltungen bzw. Einrichtungen anzuzeigen. Allerdings wird dieser Plugintyp auch nur im Veranstaltungs- und Einrichtungskontext geladen. Das heißt auf der Startseite oder im globalen Adminbereich steht es nicht zur Verfügung und es lassen sich dort auch keine Konfigurationen vornehmen. Deshalb kann es nötig sein, dass ein Plugin mehrere Typen besitzt.

Die verschiedenen Typen werden durch Interfaces repräsentiert und können durch die jeweilige Plugin-Klasse implementiert werden. Aufgrund dieser verschiedenen Plugintypen kann es sinnvoll sein mehrere statt nur ein einziges Plugin für eine neue Funktion zu implementieren.

Die Vorteile einer solchen eher modularisierteren Implementation durch mehrere Plugins sind die bessere Übersichtlichkeit, der Programmcode kann strukturierter aufgebaut werden und eine Wiederverwendung ist gegebenenfalls einfacher.

Ein Beispiel hierfür bietet das Schwarze Brett-Plugin[9]. In der Manifestdatei sind zwei Plugins hinterlegt.

```
1 pluginname=SchwarzesBrettPlugin
2 pluginclassname=SchwarzesBrettPlugin
3 pluginclassname=SchwarzesBrettWidget
4 origin=UOL
5 version=4.5.1
6 description=Globales Schwarzes Brett fuer Kleinanzeigen mit Kategorien (Gesuche/Angebote)
7 studipMinVersion=4.5
```

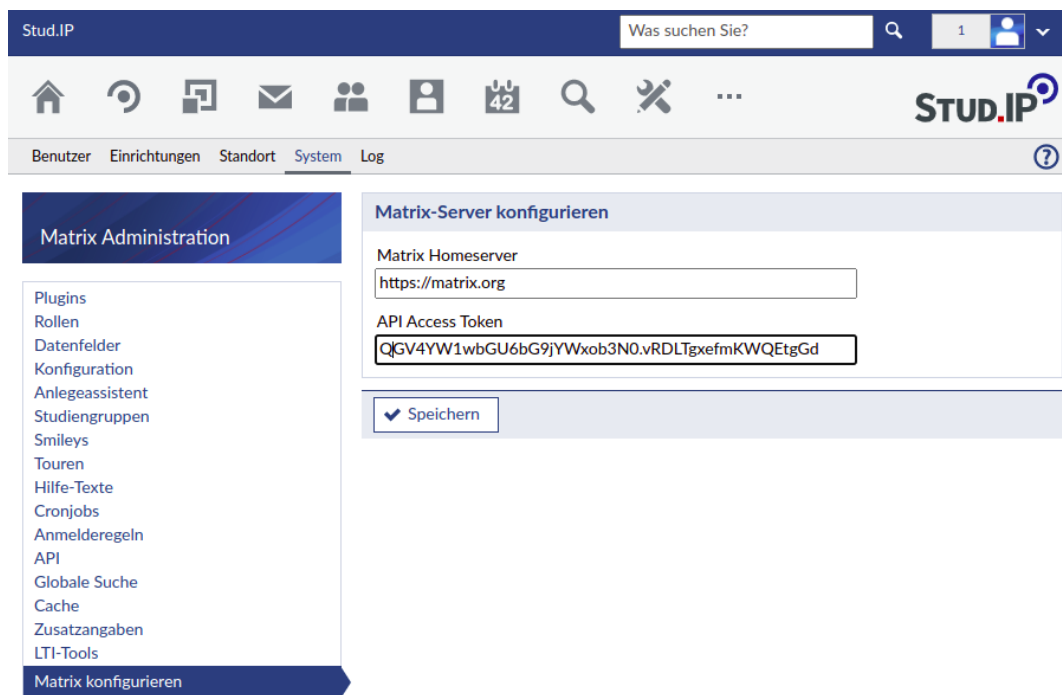
Mehrere Plugins können im Manifest zusammengefasst werden. Für das hier implementierte Beispiel könnte man folgende Funktionalitäten in separate Plugins auslagern: Adminkonfiguration und NotificationCenter.

Aus Einfachheitsgründen und der besseren Wartung entsteht die Matrix-Anbindung allerdings als ein einziges Plugin. Hat man das Plugin als Administrator oder Root installiert und aktiviert, so hat man Zugriff auf die Konfigurationsseite (siehe Abbildung 4.1).

Damit man aus Stud.IP heraus Anfragen an die API stellen kann muss zum einen der Matrix-Homeserver und zum anderen ein API-Access-Token eingetragen werden. Damit ist die Grundkonfiguration des Plugins abgearbeitet und es können Räume aus Stud.IP erstellt werden.

## 4.12 Robustheit, Ausfallsicherheit

Die Idee einer gewissen Ausfallsicherheit ohne den Verlust von Nachrichten, sollte der Matrix-Server mal nicht ansprechbar sein, sieht so aus, dass man in der Stud.IP Datenbank eine Tabelle anlegt, in der die noch nicht übertragenen Nachrichten gespeichert werden, bis sie zu Matrix gesendet wurden. Realisiert werden kann dies mit Hilfe eines regelmäßig ablaufenden Cronjobs, der prüft, ob noch Nachrichten in der Tabelle liegen, die nicht an Matrix geschickt wurden. Ist



**Abbildung 4.1:** Matrix: Adminkonfiguration

dies der Fall, versuche sie zu Matrix zu senden und lösche sie bei Erfolg. Ansonsten versuche es in 5 Minuten erneut und speicher alle neuen Blubber Nachrichten in der Datenbank.



# Kapitel 5

## Evaluation

In diesem Kapitel soll kurz auf die Chancen und die Probleme eingegangen werden, die existieren oder aufgefallen sind.

### 5.1 Chancen

Wie im Motivationskapitel bereits angesprochen sind viele Studierende unzufrieden mit der hohen Nutzung von privaten Kommunikationswerkzeugen für ihr Studium. Dies bietet vermutlich die größte Chance, wenn es gelingt die Studierenden von ihren privaten Nachrichtendiensten, wie z.B. WhatsApp, Telegram oder Discord, hinzubewegen zu einem universitätsinternen Chat-System, bei dem Server, sowie Client hausintern gehostet werden.

Somit ließe sich für die Studierenden Privates von Studium und Arbeit trennen. Weiterhin liegt es natürlich in der Hand eines jeden selbst, ob man einen entsprechenden Client auf seinem Smartphone installieren möchte oder nicht.

Neben den Studierenden bietet diese Chat-Lösung natürlich auch Vorteile für Lehrende und auch Mitarbeiter. Die Kommunikation wird im Gegensatz zu Blubber um ein vielfaches vereinfacht und beschleunigt, da man unabhängig von Stud.IP ist und sich nicht erst Anmelden, in die Veranstaltung gehen und Blubber auswählen muss, bis man eine Nachricht schreiben kann.

### 5.2 Probleme

Augenfällige Probleme sind unter anderem die Bereitstellung eines weiteren Chat-Clients für die Studierenden und die damit nicht gesicherte Akzeptanz dieses Ansatzes.

Weiterhin ist es fraglich, ob eine solche Matrix-Anbindung an Stud.IP wirklich mehr genutzt wird, als Blubber bisher. Es wäre natürlich wünschenswert, allerdings wollen viele Studierende und Lehrende vielleicht nicht noch einen weiteren Client nutzen müssen.

Eine weitere Herausforderung ist die Vermeidung von Verwirrung und die klare Abgrenzung von Blubber in Stud.IP und Matrix. Auch, wenn Nachrichten und Ereignisse untereinander versendet werden können, muss den Nutzern eindeutig klar sein, dass es sich um zwei separate Systeme handelt, die nicht austauschbar sind.



## Kapitel 6

# Ausblick

Das hier entwickelte Plugin ist in seinem derzeitigen Zustand noch nicht für den Praxiseinsatz bereit, auch wenn es mit dieser Intention begonnen wurde. Wenn es jedoch Bedarf dafür aus der Studierendenschaft und der Community gibt, steht einer potentiellen Nutzung in Zukunft nichts im Wege.

Sollte es zu einem Einsatz in der Praxis kommen, so lohnt es sich unter Umständen einmal die Bridge-Lösungen von Matrix näher zu betrachten und zu testen, da die Nutzer dann evtl. einen ihnen bereits bekannten Chat-Client für die Kommunikation über Matrix nutzen könnten.

Die Aufgrund von Zeitmangel ausgelassenen Anforderungen bieten viel Raum für Verbesserungen sowie Erweiterungen und könnten das kollaborative Arbeiten an der Universität Osnabrück ein Stück einfacher gestalten.



# Literaturverzeichnis

- [1] BIGBLUEBUTTON: *API Dokumentation: create.* – <https://docs.bigbluebutton.org/dev/api.html#create>
- [2] BIGBLUEBUTTON: *Open Source Virtual Classroom Software.* – <https://bigbluebutton.org/>
- [3] GITHUB: *An LDAP3 auth provider for Synapse.* – <https://github.com/matrix-org/matrix-synapse-ldap3>
- [4] GITHUB: *Element - A glossy Matrix collaboration client for the web.* – <https://github.com/vector-im/element-web/>
- [5] GITHUB: *Etherpad: A modern really-real-time collaborative document editor.* – <https://github.com/ether/etherpad-lite>
- [6] GITHUB: *Stud.IP Plugin: BigBlueButton und AdobeConnect Meetings.* – <https://github.com/virtUOS/studip-meeting>
- [7] GITHUB: *Stud.IP Plugin: Etherpad.* – <https://github.com/luniki/studip-plugin-studipad>
- [8] GITHUB: *Stud.IP Plugin: Moodle Konnektor für Stud.IP.* – <https://github.com/elan-ev/studip-moodle-connect>
- [9] GITLAB: *Stud.IP Plugin: Schwarzes Brett.* – <https://gitlab.studip.de/uol/plugins/SchwarzesBrettPlugin>
- [10] JSON:API: *Latest Specification (v1.0).* – <https://jsonapi.org/format/>
- [11] MATRIX: *Bridges.* – <https://matrix.org/bridges/>
- [12] MATRIX: *Client Server API.* – <https://matrix.org/docs/guides/client-server-api>

- 
- [13] MATRIX: *Frequently Asked Questions*. – <https://matrix.org/faq/>
  - [14] SCHULMEISTER, R.: *Lernplattformen für das virtuelle Lernen: Evaluation und Didaktik*. München : Oldenbourg Wissenschaftsverlag, 2005
  - [15] STUD.IP: *Entwicklerdokumentation: Die Stud.IP-Plugin-Schnittstelle*. – <https://hilfe.studip.de/develop/Entwickler/PluginSchnittstelle>
  - [16] STUD.IP: *Entwicklerdokumentation: JSONAPI*. – <https://hilfe.studip.de/develop/Entwickler/JSONAPI>
  - [17] STUD.IP: *Entwicklerdokumentation: Notifications*. – <https://hilfe.studip.de/develop/Entwickler/Notifications>
  - [18] STUD.IP: *Nutzerdokumentation: Blubber*. – <https://hilfe.studip.de/help/4.1/de/Basis/InteraktionBlubber>
  - [19] STUD.IP: *Nutzerdokumentation: Meetings*. – <https://hilfe.studip.de/help/4.4/de/Basis/Meetings>
  - [20] STUD.IP: *Stud.IP-Portal: Organisieren, Verwalten Lernen, Lehren*. – <https://studip.de/>
  - [21] TU CLAUSTHAL: *TU Clausthal: Moodle (RZ-Dokumentationen)*. – <https://doku.tu-clausthal.de/doku.php?id=multimedia:moodle:start>
  - [22] W. WEISFLOG, A. B.: Ein studentischer Blick auf den Digitalen Turn - Auswertung einer bundesweiten Befragung von Studierenden für Studierende. In: *Hochschulforum Digitalisierung* 54 (2020)

## Anhang A

# Umfrageergebnisse zur Anforderungsanalyse der Stud.IP-Matrix-Integration

Was wäre Ihnen bei der Anbindung von Matrix (Element) an Stud.IP (Blubber) wichtig? Welche Anforderungen oder Funktionen sollten Ihrer Meinung nach erfüllt werden?

Neben den normalen Matrix Funktionen (Chats mit anderen Personen und Chaträume, im besten Fall basierend auf Kursen) wäre auch wünschenswert: - eine Benachrichtigungsfunktion (Z.B. über Rückmeldefrist für das Studium, Uni schließt wegen Sturm), bei der die Lesenden nicht antworten und kommentieren können. - eine Sprechstundenfunktion, bei der man für eine kurze Zeit einen 1-zu-1 Chat mit einer Person führt und den Chat danach auch automatisch verlässt. Andere Personen sind in der Zeit in einer Warteschlange. - Eine Support-Hotline Funktion wo man einer Gruppe von Personen erreichen kann, um ein Anliegen zu besprechen und der Chat nach dem das Anliegen erledigt ist auch wieder beendet wird, bzw nur der Kunde aus dem Chat dann entfernt wird.

1 response

Ich nutze StudIP nicht

1 response

1. Mit wenigen Klicks kleine Arbeitsgruppen bilden können 2. Übernahme des Stud.IP-Profilbildes (Wenn keines eingestellt ist, ein automatisch erzeugtes verwenden, um irgendein visuelles Alleinstellungsmerkmal zu haben) Meine persönliche Meinung ist, ein Profilbild gehört zur Netiquette bzw. sollte sogar Voraussetzung für die Teilnahme sein. Es kann auch ein Naturfoto oder Legomännchen sein, muss ja kein Portraitfoto sein 3. Es sollte sehr, sehr einfach möglich sein, den Status (werde ich online angezeigt oder nicht?) einzustellen bzw. diesen zu verbergen (so dass andere nicht sehen können, ob ich online bin oder nicht).

1 response

- Das es egal ist ob ich in Matrix oder Stud.IP jemandem eine Nachricht schicken, diese also in beiden "Systemen" angezeigt wird und wenn gelesen, auch nicht weiterhin als neue Nachricht im anderen System angezeigt wird - ähnliches gilt für Räume, wenn ich also in Blubber einen Raum erstelle sollte dieser auch (wenn gewollt) in Matrix erscheinen - Stumm schalten von bestimmten (Veranstaltungs-)Räumen - Das in beiden Systemen ein ähnlicher Funktionsumfang verfügbar ist (wenn einer mehr kann wäre das auch okay, aber nicht das eine System kann das und das andere kann das)

1 response

Matrix sollte sich im visuellen Design deutlich von StudIP abgrenzen, sodass dort ein "neuer Raum" entsteht, der nicht mit dem LMS assoziiert wird, um eine "non-threatening" Umgebung zur Diskussion zu schaffen. Matrix sollte nicht in Blubber eingebettet werden, sondern weiterhin als externes Programm nutzbar sein. Außerdem finde ich Aspekte der Usability, wenige Klicks und ein modernes, minimalistisches und schönes Erscheinungsbild wichtig.

1 response

Die Frage ist etwas verwirrend. Soll Matrix oder Element angebunden werden? Annahme Matrix: Der Chat in Stud.IP sollte auf einen Kanal in Matrix gemappt werden und Stud.IP sollte als regulärer Client agieren, nicht Synapse oder Dendrite ersetzen. Der Chat in Stud.IP sollte möglichst einfach sein. Die Nutzer sollten die selben (LDAP)-Nutzer sein. Annahme Element: Lieber einen raum direkt verlinken als Element per IFrame einbinden.

1 response

Übernahme von Rechten und Rollen, Kompatibilität von Meldungen in Blubber und Element, Attachments aus elements müssen auch in blubber ankommen und umgekehrt, Emoticons müssen plattformübergreifend übertragbar sein, abgesandte Nachrichten müssen bearbeitbar und löscher sein und plattformübergreifend bei Modifikationen synchronisiert werden

1 response

Räume, die relevant für das Studium sind (z.B. für Veranstaltungen) sollten in Stud.IP an der passenden Stelle verlinkt sein; Einbindung des Element-Icons, damit der schnelle Zugriff möglich ist; Einbindung so, dass es für Dozent\*innen nicht nervig ist. Braucht es Blubber noch, wenn Kommunikation zielgerichteter über Elements läuft?

1 response

- Automatische Erstellung von Matrix-Räumen für jede verknüpfte Stud.IP-Veranstaltung - Übernahme der Teilnehmerlisten - Spiegeln wichtiger Ereignisse des Stud.IP-Kurses in den Matrix-Raum (z.B. "Neue Datei hochgeladen") - Eventuell: Mehrere Channels pro Kurs, Zusammenfassung dieser Channels als Community

1 response

- Wichtige Ereignisse auf dem Stud.IP-Kurs sollten in Matrix angezeigt werden - Direkter Zugriff auf den Matrix-Raum von Stud.IP aus (nicht neu einloggen) - Teilnehmer aus dem Stud.IP-Kurs automatisch auch in Matrix-Raum

1 response

Ich kenne Blubber nicht, aber ich denke das verlinken von Inhalten in Stud.IP sollte recht natürlich funktionieren. Umgekehrt auch das einbinden von Inhalten im Matrix-Chat in Stud.IP.

1 response

Es sollte transparent sein, was wo gepostet wird. Blubber sollte threaded discussions behalten. In Matrix sollte klar sein, wenn etwas zu Blubber gepostet wird.

1 response

Community-Funktion sollte enthalten sein; Anlegen von Channels aus der jeweiligen Stud.IP-Veranstaltung heraus; feingranulare Berechtigung für Studierende

1 response

Nur ein Login, Überblick über neue Nachrichten beim einloggen in StudIP, Benachrichtigungen auf dem Handy über neue Nachrichten
1 response
Klare Übersicht (Nachrichten und Bedienung), unabhängig davon, welches Betriebssystem/Browser/ScreenSize man nutzt
1 response
- Veranstaltungen und Studiengruppen auf Räume abbilden - automatisches Login von Nutzern (Doppel-Login vermeiden)
1 response
Teilnehmerlisten, Kurslisten, Direktnachrichten, Look & Feel, Portabilität. (Falls geeignet)
1 response
Mir würden die Basisfunktionen reichen. Nachrichten schreiben, empfangen. Räume.
1 response
Gute Integration, weg von Blubber
1 response
zeitnahe Synchronisierung
1 response



Was trifft am ehesten auf Sie zu? [View options](#) ▾

☒ Ich bin in der Softwareentwicklung tätig.  
8 responses

☒ Ich bin Anwender\*in von Stud.IP und/oder Matrix/Element.  
7 responses

☒ Ich bin in der Konzeption und/oder Beratung tätig.  
5 responses



# Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Osnabrück, August 2021