UNIVERSITY OF APPLIED SCIENCES RAPPERSWIL
DEPARTMENT OF COMPUTER SCIENCE

BACHELOR THESIS

XMPP-Grid Broker

Authors:
Fabian HAUSER and
Raphael ZIMMERMANN

Advisor:
Prof. Dr. Andreas Steffen

External Co-Examiner:
Dr. Ralf Hauser,
PrivaSphere AG

Internal Co-Examiner:
Prof. Dr. Thomas Bocek

Spring Term 2018

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Task Description

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Bachelorarbeit 2018

XMPP-Grid Broker

Studenten: Fabian Hauser, Raphael Zimmermann

Betreuer: Prof. Dr. Andreas Steffen

Ausgabe: Montag, 19. Februar 2018

Abgabe: Freitag, 15. Juni 2018

Einführung

Die IETF Security Automation and Continuous Monitoring (SACM) Working Group verfolgt eine Publish-Subscribe Architektur [1] basierend auf dem XMPP Protokoll [2] mit dem potentiell Hunderte oder Tausende von Endpunkten (PCs oder IoT Geräte) in real-time Sicherheitsinformationen in einem XMPP-Grid publizieren können. Security Information und Event Management (SIEM) Systeme können sich dann als Subscriber gezielt auf gewisse Themen (realisiert als XMPP Nodes) abonnieren.

Ein Rapid-Prototype [3] eines XMPP-Grids basierend auf einem Openfire [4] XMPP Server und einem Python Broker Script wurde am IETF 100 Hackathon Singapur im November 2017 vordemonstriert.

Für die Administration des XMPP-Grids soll eine Broker Applikation mit einem grafischen Management Interface erstellt werden. Damit sollen Themen (XMPP-Nodes) erstellt und gelöscht, sowie Owner, Publisher oder Subscriber Rechte vergeben werden können. Es soll die Möglichkeit geschaffen werden, Themen hierarchisch zu Collections [8] zusammenzufassen. Ebenfalls sollen Management-Views über verfügbare Themen, persistierte Items, Subscriber und Publisher generiert werden können.

Aufgabenstellung

- Einarbeiten in den XMPP Grid Internet Draft [1], den XMPP Standard [2], sowie die XEP-0004 [5], XEP-0030 [6], XEP-0060 [7] und XEP-0248 [8] Extensions.
- Erfassen der Requirements für einen XMPP Grid Broker.
- Erarbeiten eines Architekturkonzepts für den XMPP Grid Broker, sowie Evaluation von geeigneten Technologien für die Implementation.
- Implementation, Test und Dokumentation

Links

[1] IETF I-D *draft-ietf-mile-xmpp-grid*
https://tools.ietf.org/html/draft-ietf-mile-xmpp-grid

[2] IETF RFC 6120 *Extensible Messaging and Presence Protocol (XMPP): Core*
https://tools.ietf.org/html/rfc6120

[3] IETF 100 Hackathon *XMPP-Grid Broker Prototype*
https://github.com/sacmwg/vulnerability-scenario/tree/master/ietf_hackathon/strongSwan

[4] Openfire Homepage
https://www.igniterealtime.org/projects/openfire/

[5] XEP-0004 *Data Forms*
https://xmpp.org/extensions/xep-0004.html

[6] XEP-0030 *Service Discovery*
https://xmpp.org/extensions/xep-0030.html

[7] XEP-0060 *Publish-Subscribe*
https://xmpp.org/extensions/xep-0060.html

[8] XEP-0248 *PubSub Collection Nodes*
https://xmpp.org/extensions/xep-0248.html

Rapperswil, 19. Februar 2018

Prof. Dr. Andreas Steffen

Abstract

Fabian HAUSER and Raphael ZIMMERMANN

XMPP-Grid Broker

The IETF Managed Incident Lightweight Exchange (MILE) working group proposes the standard "Using XMPP for Security Information Exchange" which describes how an XMPP based publish-subscribe mechanism (XMPP-Grid) can be used to exchange security-relevant information between network endpoints.

Currently, no implementation of a production-ready and platform-independent administration interface (XMPP-Grid broker) for XMPP-Grids exists.

The goal of this thesis is to design and implement an XMPP-Grid broker to configure existing controllers (XMPP servers), focusing on portability, extensibility and the aspects of security in a production environment. The broker application should enable administrators to configure XMPP-Grids in a usable and productive way.

Our proposed architecture earmarks a purely client-side web application that communicates with the controller via WebSockets or *HTTP* streams (*BOSH*). The controller is typically protected by a reverse proxy, which also hosts our application. User logins are performed using mutual TLS authentication to conform to the IETF standard draft. The resulting application is implemented in TypeScript using the Angular5 framework.

The resulting implementation enables administrators to create and configure communication topics, apprehend the underlying hierarchy and manage permissions. Additionally, persistent items of topics can be inspected and published.

The XMPP-Grid broker implementation incorporates the specified functionality, resulting in a robust, ready-to-use solution. A few supplementary helpers, such as autocomplete or filtering, could not be implemented due to limitations of the underlying XMPP standards. In the future, it will be possible to realise these improvements by enhancing the related XMPP standards or by specifying and implementing proprietary protocols.

Management Summary

Motivation

The IETF standard draft "Using XMPP for Security Information Exchange" describes how devices can exchange security-relevant information within a network, a so-called "XMPP-Grid", using the widely adopted messaging protocol "XMPP". Such security-relevant information can be used to take protective measures automatically, e.g. to block devices running outdated software.

An administration interface, referred to as XMPP-Grid broker in the standard draft, is used to configure an XMPP-Grid. Currently, no such XMPP-Grid broker exists that is production ready and cross-platform.

Project Goals, Approach

The goal of this thesis is to engineer an XMPP-Grid broker that allows administrators to configure XMPP-Grids in a straight-forward and productive way. The resulting application should depend only on the underlying standards and not on specific implementations to support the further standardisation process.

A comprehensive analysis of the underlying standards is carried out in the first part of this thesis. Particular attention is given to portability, extensibility and aspects of security in a production environment. This analysis enables a systematic selection of possible architecture options to be executed in the form of architectural decisions.

Results

The resulting implementation enables administrators to create and configure communication topics, apprehend the hierarchy of the underlying XMPP-Grid and manage the permissions of network participants. Additionally, persistent messages can be inspected and published on communication topics.

The XMPP-Grid broker is implemented as an Angular5 web application that connects directly from the web browser to the XMPP-Grid. Secure communication is assured by the use of mutual authentication via TLS.

Prospects

The XMPP-Grid broker implementation incorporates the specified functionality, resulting in a robust, ready-to-use solution. A few supplementary helpers, such as autocomplete or filtering, could not be implemented due to limitations of the underlying XMPP standards.

In the future, it will be possible to realise these improvements by enhancing the related XMPP standards or by specifying and implementing proprietary protocols.

Acknowledgements

We would like to thank our advisor, Prof. Dr. Andreas Steffen, for his continuous support and helpful comments.

Tobias Brunner provided us with valuable feedback on our architecture and introduction section.

Furthermore, we would like to thank Andrea Jurt Massey for her feedback regarding writing and language use.

Contents

Task Description iii

Abstract iv

Management Summary v

Acknowledgements vi

Contents vii

1 Introduction 1
- 1.1 Motivation . . . 1
  - 1.1.1 Present Situation . . . 1
  - 1.1.2 Problem and Vision . . . 1
- 1.2 Scope Delimitation . . . 2

2 Analysis 3
- 2.1 Terminology . . . 3
- 2.2 Technical Background . . . 3
  - 2.2.1 XMPP (Extensible Messaging and Presence Protocol) . . . 3
  - 2.2.2 Relevant XMPP Extensions . . . 4
- 2.3 Domain Analysis . . . 5
  - 2.3.1 IETF Standard Draft: Using XMPP for Security Information Exchange . . . 5
  - 2.3.2 Domain Specific Language . . . 6
- 2.4 Requirements Analysis . . . 6

3 Concept 7
- 3.1 Architecture . . . 7
  - 3.1.1 Actors and Context . . . 7
  - 3.1.2 Architectural Style . . . 7
  - 3.1.3 Platform . . . 8
  - 3.1.4 Authentication and Connection Security . . . 10
  - 3.1.5 Concurrency, Scalability and Performance . . . 11
- 3.2 Wireframes . . . 11
- 3.3 Security Considerations . . . 11
  - 3.3.1 The XMPP Protocol . . . 11
  - 3.3.2 Client Security . . . 12
  - 3.3.3 Server Security . . . 13
- 3.4 Security Risk Mitigation . . . 14
  - 3.4.1 Development . . . 14
  - 3.4.2 Client Security Checklist . . . 14
  - 3.4.3 Operations Security . . . 15

4 Implementation and Testing **16**
4.1 Development Setup . . . 16
4.2 Encountered Problems . . . 17
4.2.1 Multiple Administrators . . . 17
4.2.2 Audit Trails . . . 17
4.2.3 Logout . . . 17
4.2.4 XMPP or XEP Standards . . . 18
4.2.5 Openfire XMPP Server . . . 19
4.2.6 Limited Error Handling . . . 19
4.3 Code Quality . . . 20
4.4 Testing . . . 20
4.5 Documentation . . . 21

5 Discussion and Conclusion **23**
5.1 Achieved Result . . . 23
5.1.1 Implemented Requirements . . . 23
5.1.2 Architecture . . . 24
5.1.3 Implementation . . . 25
5.2 Lessons Learned . . . 26
5.2.1 Project Course . . . 26
5.2.2 Architectural Decisions . . . 26
5.2.3 Development, Frameworks and Tooling . . . 27
5.2.4 Standards . . . 27
5.3 Future work . . . 28
5.4 Conclusion . . . 28

Bibliography **III**

List of Figures **IV**

List of Tables **V**

Glossary **VI**

Appendices **IX**
A.1 Project Plan . . . X
A.2 Development Guide . . . XXV
A.3 Architectural Decisions . . . XXVI
A.4 Time Accounting . . . XLIII
A.5 Meeting Minutes . . . XLIV
A.6 Requirements . . . LXVI
A.6.1 Authentication . . . LXVI
A.6.2 List Topics and Collections . . . LXVII
A.6.3 Create a New Topic . . . LXVIII
A.6.4 Create a New Collection . . . LXVIII
A.6.5 Delete an Existing Topic . . . LXIX
A.6.6 Delete an Existing Collection . . . LXIX
A.6.7 Manage Topic/Collection Subscriptions . . . LXIX
A.6.8 Manage Topic Affiliations . . . LXX
A.6.9 Manage Persisted Items of a Topic . . . LXXI
A.6.10 Manage Subscription Requests (optional) . . . LXXII
A.6.11 Validate Controller Configuration (optional) . . . LXXII

A.7 Wireframes LXXIII
A.8 Comparison of XMPP Server and Libraries LXXVIII
A.8.1 Server LXXVIII
A.8.2 Libraries LXXIX
A.9 Personal Reports LXXXI
A.9.1 Raphael Zimmermann LXXXI
A.9.2 Fabian Hauser LXXXI

Declaration of Authorship **LXXXII**

Chapter 1

Introduction

"Every accomplishment starts with the decision to try."
— unknown

In this chapter, we present the motivation and legitimisation of our thesis and highlight the scope delimitations.

1.1 Motivation

In this first section, we shall legitimate this thesis and explain the value and applicability of our proposed solution.

1.1.1 Present Situation

The Internet Engineering Task Force (*IETF*) standard draft *Using XMPP for Security Information Exchange* [8], as summarised in section *2.3.1 IETF Standard Draft: Using XMPP for Security Information Exchange*, defines a protocol to exchange security-relevant information between network endpoints. The draft was created by the Managed Incident Lightweight Exchange (*MILE*) working group to support computer and network security incident management.

Hereafter, we refer to this *IETF* standard draft as *XMPP-Grid standard*.

To demonstrate the viability of the *XMPP-Grid standard* a rapid prototype was developed in November 2017 [37].

1.1.2 Problem and Vision

Currently, no implementation of the *XMPP-Grid standard* management functionality exists that is ready for production use regarding usability and security.

To solve this problem, a graphical interface with bindings to a suitable *broker* must be proposed and implemented. The interface should permit network administrators to manage and review *topics*, *persisted items* and *platforms*. Additionally, *consumers* and *providers* permissions of *topics* and *platforms* must be manageable.

The graphical interface supports administrators to better understand the underlying hierarchy and affiliations of *topics*, enabling them to assess security implications. As the interface uses familiar terminology known to an administrator, no in-depth understanding of the underlying *XMPP* technology is required to configure and comprehend an *XMPP-Grid*. Finally, the interface also provides better usability than existing command line interfaces and *XMPP* configuration software, which leads to fewer configuration mistakes and improved efficiency.

We hope that with the help of our implementation the *IETF* draft "Using *XMPP* for Security Information Exchange" will become an established security standard used in practical industry applications.

1.2 Scope Delimitation

As described in the *Task Description*, the focus of this thesis is on the evaluation, design and implementation of the *XMPP-Grid broker*. Adding missing functionality or fixing complex bugs in existing server or client implementations are beyond the scope of this thesis.

Chapter 2

Analysis

"Without requirements or design, programming is the art of adding bugs to an empty text file."
— Louis Srygley

2.1 Terminology

Taking into account that developers and operators of security reporting systems are the intended audience for this thesis, we mostly use the Security Automation and Continuous Monitoring (SACM) terminology [5] and thereby follow the same guidelines as the *XMPP-Grid standard* [8].

2.2 Technical Background

The following sections introduce the *XMPP* protocol including its terminology and summarise the relevant extensions (*XEPs*) used by the *XMPP-Grid standard*.

2.2.1 XMPP (Extensible Messaging and Presence Protocol)

The Extensible Messaging and Presence Protocol (in short *XMPP*, formerly known as *Jabber*) is an open protocol that enables the near-real-time exchange of small data between any network endpoints, hereafter called *platforms* [31]. While originally designed as an instant messaging (IM) protocol, *XMPP* can be used for a wide range of data exchange applications [30].

XMPP is made of small building blocks defined in the core protocol [31] and numerous extensions called *XEPs* [33]. The core specifies how encrypted communication channels must be established, how *XML stanzas* are exchanged and errors are handled.

The core is comprised of *XML* streams, error handling and functionality for establishing encrypted communication channels. Additional functionality such as *service discovery* [18] and *publish-subscribe* [23] are defined in separate extensions.

Although *XMPP* supports peer-to-peer communication, it is often used in a traditional client-server architecture. A client (*platform*) can send data to any addressable entity (any other *platforms*) using *Jabber* identifiers, hereafter called *JID*. If the receiving *JID* has a different domain than the current server (*controller*), the message is forwarded to the *XMPP* server responsible for this domain. [31]

The data exchanged over *XMPP* is in the *XML* format, which makes the protocol structured and extensible, but leads to some protocol overhead. An *XMPP* client

communicates with the server over unidirectional data streams, that are basically long-lived *TCP* connections. A client opens a channel to the server over this connection, and the server reacts by opening a connection in the opposite direction. In both streams, an XML document is opened after the connection is established (i.e. with `<stream>` XML tags). During the conversation, an arbitrary amount of *stanzas* (specified XML child elements) are written to the stream. Before a connection may be terminated, the root element is closed (i.e. `</stream>`) and both streams form valid XML documents [31, 24].

The core *stanza* types are *messages* (`<message/>`), *presence* (`<presence/>`) and *info/query* (`<iq/>`). *Messages* can contain arbitrary data similar to email but are optimised for immediate delivery. *Presence stanzas* deal with network availability and the propagation of user presence information. An *info/query stanza* consists of a request and response (similar to the GET and POST *HTTP* methods), which is used for feature negotiation, configuration and general information exchange. Because of these coarse semantics, *XMPP* provides a generalized communication layer [31, 30].

Figure *2.1* illustrates an example setup with two servers and three clients.

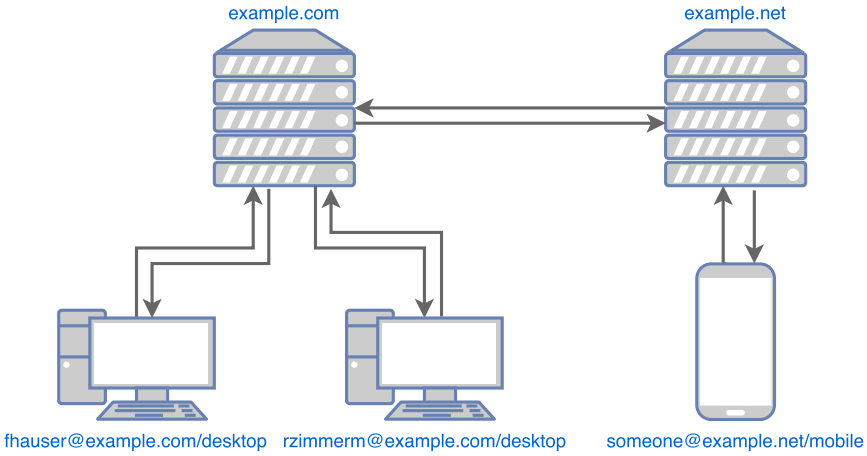


FIGURE 2.1: Two *XMPP* domains (servers), one with two users and one with a single mobile user.

2.2.2 Relevant XMPP Extensions

The *XMPP-Grid standard* is based on multiple *XEPs*, most notably the *publish-subscribe* extension. In this section, we give an overview of the most relevant used *XEPs*.

XEP-0004: Data Forms is a flexible protocol that can be used in workflows such as service configuration. The protocol provides form processing, common field types and extensibility mechanisms. [11]

XEP-0030: Service Discovery enables entities to discover information about the identity and capabilities of other entities, e.g. whether the entity is a server or not, or items associated with an entity, e.g. a list of *publish-subscribe* nodes. [18]

XEP-0059: Result Set Management allows entities to manage the receipt of large result sets, e.g. by paging through the result or limiting the number of results. *result set management* is often desired when dealing with large dynamic result sets, as

from service discovery or publish-subscribe, and when time or other resources are limited. [26]

XEP-0060: Publish-Subscribe

The *publish-subscribe* extension, hereafter referred to as *pubsub* or *broker*, enables *XMPP* entities (*providers*) to broadcast information via *topics* to subscribed entities (*consumers*). [23]

Nodes, hereafter referred to as *topics*, are the communication hubs. Entities can create *topics* and configure them, e.g. set up subscription timeouts or limit publishing and subscription rights. The configuration mechanism is based on data forms (XEP-0004). An *XMPP* server *may* restrict *topic* creation to certain entities, which means that possibly not every *XMPP*-Server that supports *publish-subscribe* also implements this feature [6].

The protocol defines a hierarchy of six affiliations of which only the implementation of owner and none is *required*. Implementing the remaining four affiliations is *recommended*. An owner of a *topic* can manage the subscriptions and affiliations of other entities associated with a given *topic*.

To simplify the creation of *topics*, *pubsub* defines five *topic* access models (node access models) that *should* be available: open, presence, roaster, authorize and whitelist. The open model allows uncontrolled access while presence and roaster are specific for IM. Using the authorize model, the owner has to approve all subscription requests. The whitelist model enables the owner to maintain a list of entities that are allowed to subscribe.

2.3 Domain Analysis

2.3.1 IETF Standard Draft: Using XMPP for Security Information Exchange

This *IETF* standard draft describes how the *XMPP* protocol and its extensions can be used for the exchange and distribution of security-relevant information between network devices.

One of the primary motivation for using *XMPP* for this task is the fast propagation of such security-relevant data. Using *XMPP* for such a task also comes with its downsides. Most notably, because the *XMPP* server (*broker*/*controller*) is the central configuration component in charge of managing access permission, its compromisation has serious consequences.

The standard describes a trust model, a thread model as well as specific countermeasures, e.g. to use at least *TLS* 1.2. These countermeasures also define restrictions of the *XMPP* protocol and its extensions, e.g. by limiting the *topic* access models of *pubsub* to whitelist and authorized only [8].

Information Exchange Format

The *XMPP-Grid standard* states that 'although [the exchanged] information can take the form of any structured data (XML, JSON, etc.), this document illustrates the principles of *XMPP-Grid* with examples that use the Incident Object Description Exchange Format (IODEF)' [10, 8].

As IODEF is not strictly defined nor explicitly recommended by the *XMPP-Grid standard*, no specific integrations are in the scope of this thesis.

In practice, small payloads are sent over an *XMPP-Grid*, usually containing external pointers to an API that provides more comprehensive data (see appendix *A.5 Meeting Minutes*).

2.3.2 Domain Specific Language

Figures *2.2* and *2.3* present an overview of the relevant interactions and relationships between the different components as specified in the *XMPP-Grid standard* and the referenced *XEPs* (see section *2.2 Technical Background*).

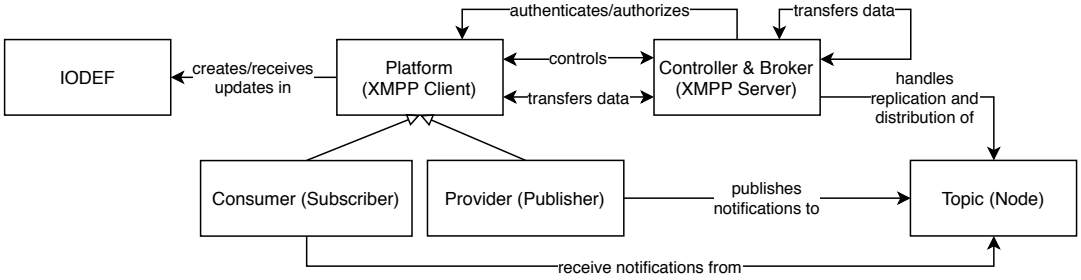


FIGURE 2.2: Domain specific language of the *XMPP-Grid standard*.

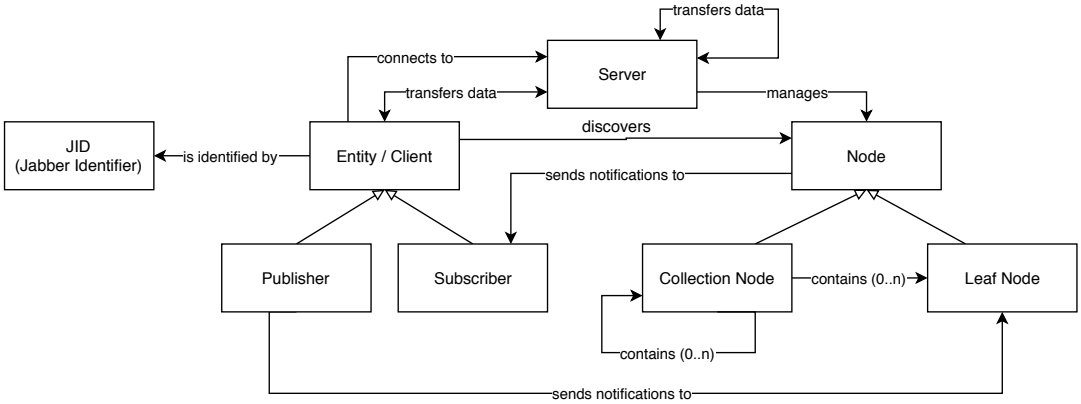


FIGURE 2.3: Domain specific language of used *XMPP XEPs*.

2.4 Requirements Analysis

We collected the functional requirements in the form of user stories. User stories are an established and widespread concept for describing and managing requirements in agile software projects. In comparison to traditional tools for requirement analysis, user stories are more concise, leaving more space for change. [41]

We also created user stories for non-functional requirements. Additional non-functional requirements can be added during the project in the form of constraints. [41]

In the early phase of the project, we collected an initial set of user stories in collaboration with Prof. Dr. Steffen. This initial set covered the creation and deletion of *topics* as well as granting publish and subscribe privileges. All user stories are listed in appendix *A.6 Requirements*.

After setting broad priorities, Prof. Dr. Steffen approved the initial set of user stories that then served as the basis for the architectural concept.

Chapter 3

Concept

"Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away."
— Antoine de Saint-Exupery

3.1 Architecture

In this section, we present the architecture and fundamental architectural decisions of the *XMPP-Grid broker* application. All architectural decisions we took are fully documented in appendix *A.3 Architectural Decisions*.

We illustrate the concepts and structures using the C4 Model for Software Architecture [7].

3.1.1 Actors and Context

The context diagram pictured in figure *3.1* shows the actors and surrounding systems that are given for the *XMPP-Grid broker*, as described in the *Task Description*.

One or more administrators manage the *XMPP-Grid* by adding or removing *platforms* and configuring *topics*. To minimize the required work and reduce the error-proneness, administrators interact with the *XMPP-Grid broker*.

The *XMPP-Grid broker* configures the *XMPP-Grid*, which consists of a *controller* and *platforms*.

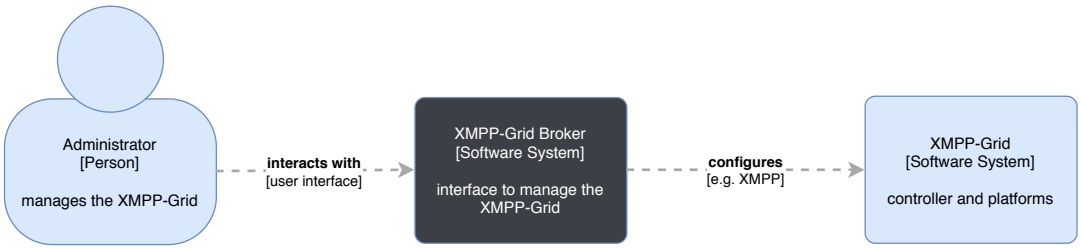


FIGURE 3.1: Architecture diagram showing the context of the *XMPP-Grid broker* application.

3.1.2 Architectural Style

To implement the *XMPP-Grid broker*, we evaluated three possible architecture styles: An *XMPP* server plug-in (e.g. extension for the Openfire *XMPP* server), an implementation using the Jabber Component Protocol [32] and an implementation acting as a regular *XMPP* client ("bot").

We decided to build an *XMPP* client/bot because, unlike a server plugin, it is not coupled to a specific *XMPP* server. In contrast to the *XMPP* component, *XMPP* clients support strong authentication mechanisms with *SASL*.

The full decision argument is documented in appendix *A.3 Architectural Decisions*.

3.1.3 Platform

The proposed *XMPP* client might be implemented in different ways: as rich client application with a command line or graphical interface as illustrated in figure *3.2*, and in the form of a web application, illustrated in figures *3.3* and *3.4*.

Rich Client Application

The idea of a rich client application is to communicate directly from a graphical desktop application with the *XMPP-Grid*, as illustrated in figure *3.2*.

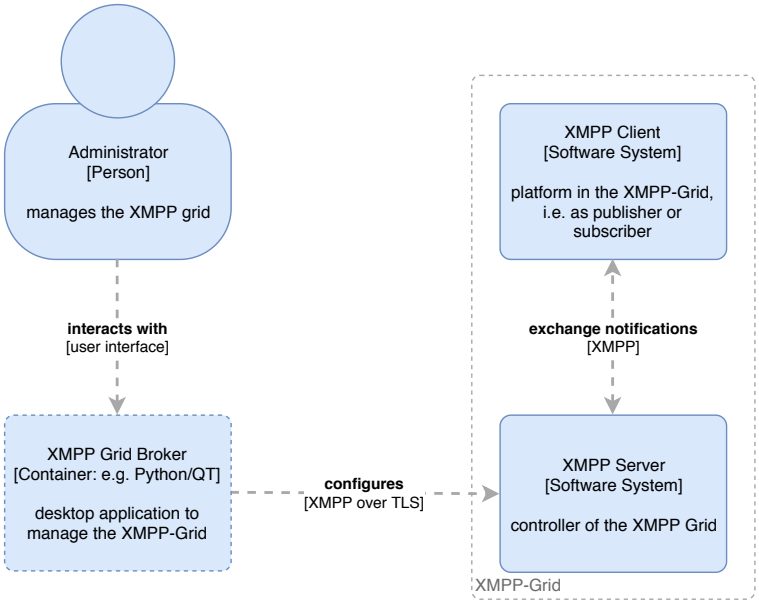


FIGURE 3.2: Architecture container diagram showing a possible rich client architecture.

Web Application

In contrast to a rich client application, a web application has the significant advantage to be easily installable and upgradable with minimal interaction on the user's side (i.e. only requires a web browser to be executed).

Therefore, we decided to implement the *XMPP-Grid broker* as web application.

To manage the *controller* from the *broker* interface, we considered either directly connecting to the *XMPP* server over WebSockets [38] or *HTTP* (*BOSH* [27]), and the indirect communication with the *XMPP* server via custom web API proxy. These topologies are illustrated in figure *3.3* and figure *3.4*.

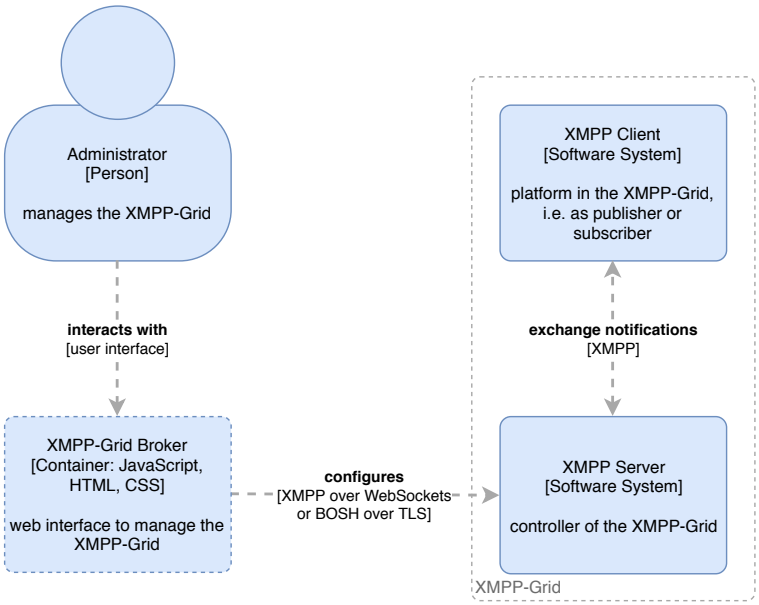


FIGURE 3.3: Architecture container diagram showing the web application topology with WebSockets or *BOSH*.

Web API Proxy

A web API proxy could be realised with a custom browser-to-proxy protocol, as implemented in the *XMPP*-FTW JavaScript library[1]. However, this approach leads to a high coupling between a concrete library and the web application.

Another approach would be the implementation of a custom WebSocket-to-*XMPP* Proxy, which allows connecting to *XMPP* servers that do not support WebSockets or *BOSH*. If such a proxy is implemented transparently, the client is not aware of the server limitations. Therefore, the client implementation is no different from direct communication with an *XMPP* server.

As the *XMPP* over WebSocket protocol differs from the normal *XMPP* protocol, a transparent proxy implementation would inevitably need to hold the connection state and implement custom keep-alive mechanisms [38].

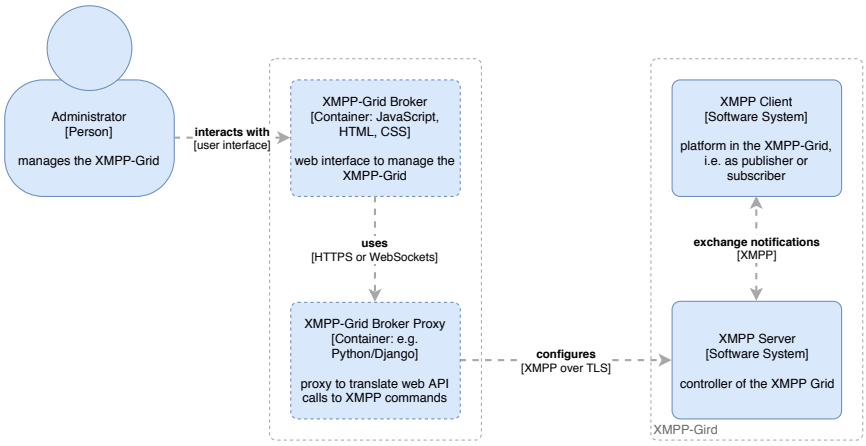


FIGURE 3.4: Architecture container diagram showing the web API proxy topology.

[1] http://docs.xmpp-ftw.org/

Implemented Web Application Topology

As described in the according architectural decision (see appendix *A.3 Architectural Decisions*), we decided on the option that connects directly via WebSockets, if possible with a fallback to *BOSH*. This topology simplifies the implementation and deployment of the application in comparison to a web API proxy. WebSockets offer stateful *TCP*-sockets to exchange data with *XMPP* servers in contrast to *BOSH*, which uses *HTTP* long polling to emulate bidirectional streams and is, therefore, less efficient [27].

To increase the *XMPP* server security, an *HTTP* reverse proxy (e.g. nginx[2]) between the client and the *XMPP* server might be added as shown in figure *3.5*. The reverse proxy might also be used to serve the web application and provide authentication (see section *3.1.4 Authentication and Connection Security*) and security features (see section *3.4.3 Operations Security*).

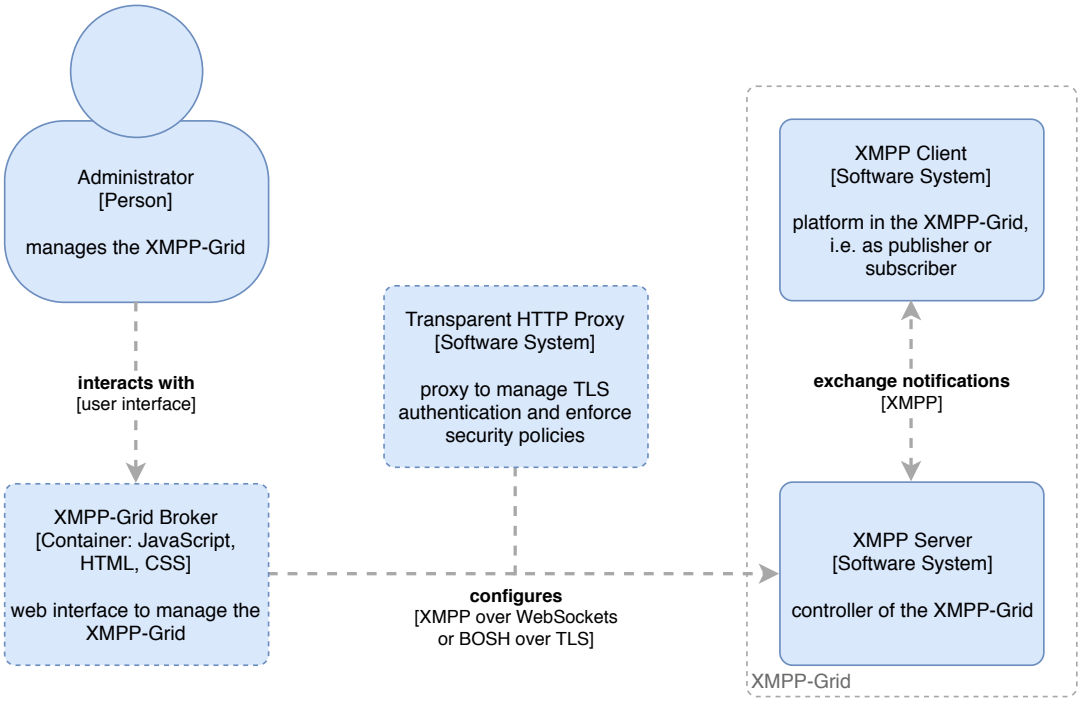


FIGURE 3.5: Architecture container diagram showing the implemented web application topology.

3.1.4 Authentication and Connection Security

XMPP uses *SASL* as authentication mechanism [31]. To authenticate against the *XMPP-Grid controller*, we decided to use the *SASL EXTERNAL* [42] mechanism whenever possible to authenticate the client.

We decided against *SASL SCRAM* [16], the alternative authentication method that is also recommended by in the *XMPP-Grid standard*. As described in the corresponding architectural decision (see appendix *A.3 Architectural Decisions*), the main reason for *SASL EXTERNAL* is its higher level of security and and its relatively simple scaling capabilities.

[2] https://nginx.org/

SASL EXTERNAL implies that the authentication takes place on a lower layer than the actual *XMPP* protocol. In our case, this implies authentication over *TLS*, i.e. X.509 user certificates as specified in RFC6120 [31].

Currently, not all *XMPP* servers that implement *BOSH* and Websockets also implement *SASL EXTERNAL* with *TLS* authentication (e.g. Openfire currently supports *TLS* authentication with *BOSH* but not with Websockets[3]). To circumvent this limitation, an *HTTP* reverse proxy (see section *3.1.3 Implemented Web Application Topology*) might be used to handle *TLS* authentication.

3.1.5 Concurrency, Scalability and Performance

As presented in the architecture above, the *broker* application communicates directly from the user's web browser with the *XMPP* server. By adopting this approach, the main scalability concern is on the *XMPP* server. This approach is also in alignment with the *XMPP* philosophy to move as much complexity as possible on the server [36].

Concurrency and scalability are the responsibility and speciality of the underlying *XMPP* server and therefore not directly relevant for the *broker* application [36].

Regarding performance, the network is the primary source of potential slowdowns. The *XMPP-Grid broker* must reduce the number of requests needed to a minimum and whenever possible execute requests in parallel. Additionally, the initial loading time of the application can be optimised.

A potentially used reverse proxy must be scaled with the number of administrators. As this number is usually rather small, no extra effort is usually required.

3.2 Wireframes

We created wireframes for most screens to visualise the initial set of user stories. They helped us to find missing requirements, most notably the support of collections. All wireframes are listed in appendix *A.7 Wireframes*.

3.3 Security Considerations

Regarding the *XMPP-Grid broker* application, there are three primary attack vectors:

Client-Side Attacks, e.g. via web browser, web browser extension or malicious software on the client operating system.

Web Server Attacks, e.g. misconfiguration or insufficient hardening.

***XMPP* Server Attacks,** e.g. misconfiguration or insufficient hardening.

Details on all these attack vectors are discussed in the following sections.

3.3.1 The XMPP Protocol

An in-depth security analysis of the *XMPP* protocol is beyond the scope of this thesis. A detailed discussion of security concerns can be found in the *XMPP* specification [31] and most XEPs [23, 34]. In this section, we highlight the most crucial security concerns relevant to this thesis.

[3] https://github.com/igniterealtime/Openfire/blob/02c22e/src/java/org/jivesoftware/openfire/websocket/OpenfireWebSocketServlet.java

Transport Security

XMPP reuses many established and standardised mechanisms to improve protocol security. By layering protocols in a strict manner (*XMPP* with *SASL* over *TLS* over *TCP*), many attack scenarios such as replaying or eavesdropping are minimised. The protocol also requires clients and servers to validate the certificates of the other party. [35, 31]

Protocol

Since *XMPP* is based on XML, it inherits some of its security implications. *XMPP* prohibits some XML features such as comments and external entity references which mitigate common attacks. [31]

The protocol itself cannot mitigate attacks where an attacker gains access to account credentials. To reduce the risk of these attack vectors best practices such as storing certificates and passwords securely must be followed.

PubSub Collection Nodes

The use of XEP-0248 PubSub Collection Nodes [34] can leak private data if not configured properly. Administrators must take great care when configuring collection nodes. The *XMPP-Grid broker* should support administrators to detect such data leaks.

3.3.2 Client Security

Because the web gives rise to many potential security concerns, above all a modern web browser is critical for client security. Legacy web browsers can not provide an adequate level of security. [15]

Most web browsers support extension mechanisms which have rather significant capabilities [29]. The usage of untrusted and uncertified browser extensions is strictly discouraged.

The same applies to the client operating system and all software installed on clients.

Authentication and Authorisation

Regarding authentication and authorisation, the *XMPP* server does most of the heavy lifting such as storing passwords and validating certificates. On the client side, the web browser does most of that work too (i.e. validating certificates).

The responsibility of a client implementation is to establish a secure channel to the *XMPP* server and warn the user if a problem occurs during this process (e.g. invalid server certificate).

Angular Framework

Using the Angular framework impacts client security significantly. Angular is built with security in mind and is adopted in the industry in security-relevant environments. Therefore, Angular receives frequent security updates and is well tested. It's unlikely that a similar security level might be reached with plain JavaScript in a reasonable implementation timespan.

On the project website, three best practices regarding security are recommended [1].

- Keep up with the latest Angular library releases.
- Don't modify your copy of Angular.
- Avoid Angular APIs marked in the documentation as "Security Risk".

We can ensure the latter two by making them acceptance criteria. Keeping current with the latest Angular releases is harder, as our work on this project is limited. To ensure that future updates can easily be applied we deviate as little as possible from the standard angular setup (e.g. by not ejecting the Webpack configuration[4]).

Keeping Angular up-to-date is of paramount importance as potential vulnerabilities (e.g. XSS) can be exploited if not patched.

Angular Content Security

Except for *persisted items*, no *XMPP* content is displayed directly but serves as the basis for rendered HTML components. To protect against malicious payloads, the received XML messages must be validated before their usage.

Persisted items can contain arbitrary content and must therefore be escaped before rendering to prevent Cross-Site Scripting (XSS) attacks.

Angular supports these measures by treating all values (except Angular templates) as untrusted by default. To prevent user-generated data to influence Angular templates, the offline template compiler is used. To fully utilise the security measures provided by Angular, official APIs must be used at all times instead of direct use of the DOM-APIs. [1]

Using Content-Security-Policy (CSP) provides additional XSS-protection mechanisms [40]. The *XMPP-Grid broker* should document an appropriate CSP that must be supported in a production environment.

3.3.3 Server Security

Authentication and Authorisation

The *XMPP* server implements most of the authentication and authorisation mechanisms used in an *XMPP-Grid broker* implementation, such as storing passwords and validating certificates.

If *BOSH* or WebSockets are used, the *XMPP* server should support most *HTTP* security features as listed in section *3.3.3 Web Server*. Additionally, the origin of WebSockets and *BOSH* requests must be verified (by either the Origin header or CORS support. [21, 39]

The web server hosting the client application has no active authentication or authorisation responsibility, except to ensure the integrity and authenticity of the application, i.e. by using *TLS*.

Web Server

To minify security concerns on the server side, we decided to keep the application files static (see *A.3 Architectural Decisions*). This allows operators to use any standard web server (e.g. nginx, Apache, etc.) to serve the client. Securing such standard web servers is common knowledge for operators and is beyond the scope of this analysis.

[4] https://github.com/Angular/Angular-cli/wiki/eject

In addition to these general best practices, we explicitly recommend the following security measures to improve client security:

- Enable Content Security Policy (CSP) [40].
- Use secure *TLS* configurations such as secure cipher suites, strictly honor cipher order, HSTS, HPKP and OCSP Stapling[14].

These recommendations should be documented in the application installation guide.

XMPP Server

XMPP server security depends on the chosen implementation and the application domain. Discussing *XMPP* server security in detail is beyond the scope of this thesis. Operators should adhere to the security recommendations of their *XMPP* server vendor and follow general security best practices as outlined by the *XMPP-Grid standard*.

3.4 Security Risk Mitigation

To mitigate the security risks as discussed in section *3.3 Security Considerations*, the measures as described in the following subsections are implemented.

3.4.1 Development

1. Conduct code reviews for all newly added code using GitHub pull requests and a security checklist (see next section)
2. Conduct an architectural analysis with an industry expert[5]
3. Automate build and release processes to minimise the time required to patch
4. Stay as close to the default Angular setup to simplify further updates
5. Avoid additional third-party dependencies whenever possible

3.4.2 Client Security Checklist

- The latest Angular-version is used
- No customizations are made to the Angular version
- No direct access to DOM-APIs
- APIs marked in the documentation as "Security Risk" are *not* used
- No usage of any methods starting with `bypassSecurityTrust`
- The client is fully Content Security Policy (CSP) compliant
- The client is fully Same Origin Policy (SOP) / Cross-Origin Resource Sharing (CORS) compliant

[5] Was carried out on 2018-04-16, see *A.5 Meeting Minutes*.

- Only compile templates with the offline template compiler (AOT)
- User input is always escaped using the mechanisms provided by the framework (eg. Angular Forms)
- *XMPP* messages are validated to contain only the specified result-types

3.4.3 Operations Security

Administrators must configure the surrounding systems correctly to mitigate certain security risks. To support administrators, we recommend the following measures.

Content Security Policy

The Content Security Policy (CSP) helps to mitigate certain types of attacks such as Cross Site Scripting (XSS) as a second line of defence [40]. The recommended values are directly documented in the project source code repository.

Verify Origin

The *XMPP* server should be configured to only accept WebSockets/*BOSH* connections from the origin of the *XMPP-Grid broker* application. The Origin header sent by the web browser must match the domain on which the *XMPP-Grid broker* application is hosted. Otherwise, connection requests must be ignored. If an *XMPP* server does not support this feature, a proxy server should be used to verify the Origin header. In the provided development setup, this security feature must not necessarily be implemented.

Chapter 4

Implementation and Testing

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."

— Martin Fowler

4.1 Development Setup

Figure *4.1* illustrates the development setup in the form of an UML deployment diagram. Developers connects from their browsers to the reverse proxy that serves the static *broker* web application. The *HTTP* connection from the client to the server is secured using mutual *TLS* authentication. The same reverse proxy also routes the *XMPP* connections. The proxy establishes a mutual authenticated *TLS* connection to the *XMPP* server. The reasons for this setup are described in more detail in section *4.2.5 Openfire XMPP Server*.

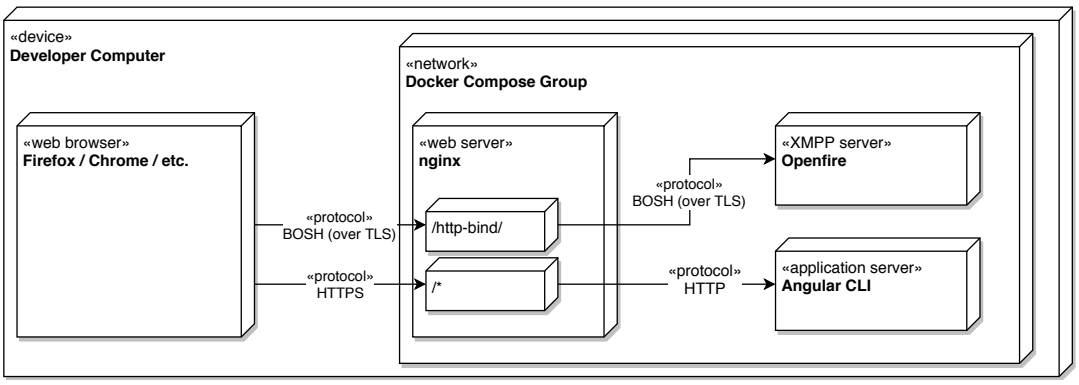


FIGURE 4.1: UML deployment diagram presenting the development setup.

As the previously described structure is not trivial, the guiding principle for our development setup was to maximise automation and minimise manual efforts. This principle is the basis for durable software. We decided on a docker and docker-compose[1] based stack that provides a correctly configured Openfire instance, a pre-configured nginx instance as well as client and server certificates. Everyday tasks such as building and testing the application and documentation were automated as bash scripts.

The efforts invested in this docker setup proved valuable when we began to write integration tests that run in the same environment.

[1] https://www.docker.com/

We deliberately decided to run unit tests outside of the docker environment as unit tests are executed more often, and the additional docker-overhead would be unnecessarily expensive. Also, debugging is more straightforward without indirections.

4.2 Encountered Problems

4.2.1 Multiple Administrators

Requirement *A.6.1.5 Multiple Administrators* states that multiple administrators should be able to access the application.

When authenticating users with *SASL EXTERNAL*, the client certificate extension field xmppAddr is interpreted as user *JID* by the *XMPP* server.

In practice, most *XMPP-Grid broker* deployments require an *HTTP* proxy in front of the *XMPP* server as security measure (see section *3.1.3 Implemented Web Application Topology*). Usually, the *HTTP* proxy can also be used to serve the *broker* application, as realised in the development setup. Such an *HTTP* proxy might also accept multiple client certificates.

If the client connects to the *XMPP* server over secure WebSockets (WSS) in combination with *SASL EXTERNAL*, the WebSocket URL must already be authenticated, as most browsers do not permit certificate selection on background requests [12]. This might be achieved by serving the *broker* from the same domain or by using client certificate policies [13].

As the proxy intercepts the *TLS* connection, it must verify the client certificate sent by the browser and establish a connection to the *XMPP* server using a client certificate as well. Therefore, the xmppAddr field of the proxy's client certificate is used by the *XMPP* server. If multiple users should be differentiated on the *XMPP* server, an *HTTP* proxy might choose different client certificates for connecting to the *XMPP* server based on the web browser's client certificate xmppAddr.

4.2.2 Audit Trails

Actions of administrators should be traceable with an audit trail according to requirement *A.6.1.6 Audit Trail*.

As outlined in section *4.2.1 Multiple Administrators*, practical deployments of *XMPP-Grid brokers* mostly use an *HTTP* proxy. The proxy can also be used to keep an audit trail of client requests. These requests can then be correlated with the query log on the *XMPP* server.

Audit trails on the client side are not trustworthy, as users might prevent trail entries by manipulating the client application. Therefore, no such mechanism was implemented.

4.2.3 Logout

Administrators should be able to terminate a session by using a logout function, as stated in requirement *A.6.1.7 Logout*.

We decided to use *TLS* client certificate authentication as part of *SASL EXTERNAL*. As a result of our decision to write a web application, the web browser authenticates users with *TLS* certificates.

Unfortunately, web browsers do not expose a standardised way to log out of a *TLS* client authenticated session [25]. To close the *TLS* session, administrators must close their browser window after using the *XMPP-Grid broker*.

4.2.4 XMPP or XEP Standards

Multiple shortcomings in the relevant *XEPs* were discovered during the realisation of the proposed architecture, that would have led to a highly inefficient implementation of some requirements.

Recursive Listing and Filtering of All Topics

Requirement *A.6.2.1 List All Topics* states that an administrator should be able to list all topics recursively.

This requirement could not be implemented efficiently, as the current *publish-subscribe XEP* does not support recursive queries of *topics*, but only root *topics* and subtopics.

Therefore, we implemented a recursive approach on the client side, that queries all root *topics* and recursively requests all subtopics to be displayed.

For the same reason, we did not implement requirement *A.6.2.8 Topic and Collection Name Filter* as searching the whole *topic* tree would require traversal on the client side. With an assumed count of approximately 1000 *topics*, this would result in large performance overhead.

Filtering and Paging of Persisted Items

Requirements *A.6.9.2 Filter Persisted Items* and *A.6.9.3 Paged Persisted Items* were built on the premise that filtering and paging of *persisted items* would be possible with the *result set management XEP*.

Retrieving multiple *persisted items* in *result set management* pages was added in version 1.12 (2008-09-03) of the *publish-subscribe XEP*. An *XMPP* server does not report, which version of the standard draft it supports.

Therefore, we could not presume an implementation of *result set management*. In fact, the Openfire *XMPP* server we used in our setup has no support for retrieving *persisted items* with *result set management*. We were still able to fetch the persisted items in pages using *service discovery*, as the *result set management* draft uses service-discovery as an example, making the server side support more likely [26].

Create and Configure Topics

We have four requirements related to the initial configuration of *topics*:

- *A.6.4.1 Override Default Topic Configuration*
- *A.6.4.2 Override Default Collection Configuration*
- *A.6.4.3 Initial topic Consumers and Providers*
- *A.6.4.4 Initial Collection Consumers*

Providing initial configuration for a *topic* is only partially possible due to limitations in the *publish-subscribe XEP*. The default configuration can be fetched, but

it must not necessarily comprise all possible configuration options of a *topic*. As managing consumers (via subscriptions) and providers (via consumers) are separate concepts from the configuration and can only be configured after a *topic* has been created, we concluded that a two-step process is more appropriate.

4.2.5 Openfire XMPP Server

As discussed in section *4.1 Development Setup*, the Openfire *XMPP* server was used in the development setup. This section details the encountered limitations while implementing the *XMPP-Grid broker*.

WebSocket SASL EXTERNAL Support

At time of writing, Openfire does not support *SASL EXTERNAL* in combination with *XMPP* over WebSockets. Therefore, the current implementation of the *XMPP-Grid broker* was developed with *BOSH*, but also supports communication over WebSockets thanks to the Stanza.io[2] *XMPP* library.

Lost Updates

When editing the configuration of a *topic*, Openfire exposes multiple fields that are mutually dependent. One example of this is the configuration of how many *persisted items* should be kept. If persisting items on a *topic* is disabled, Openfire does neither update the field nor respond with an error as specified in the standard [23, 11].

This behaviour is not user-friendly at all, as an administrator might want to change configuration options pro-actively. To circumvent this problem, a functionality to compare any changes in the new configuration of a topic after storing all changes might be implemented in the future.

Different Field Types

At time of writing, Openfire returns data form field types for some *publish-subscribe* configuration fields that deviate from the specification. Although modifing the field type is explicitly allowed by the standard [23], the usability of these fields suffers. A prominent example is the 'pubsub#node_type' field, which is presented as a text field instead of a selection.

A support request at the Openfire project regarding this issue was opened[3], which is mandatory before filing an issue in the Openfire issue tracker. However, there has been no response by the editorial deadline of this thesis.

Should the type of such fields change in the future, the flexible implementation of *data forms* in our implementation is sufficient to reflect the new form type.

4.2.6 Limited Error Handling

Running entirely in the browser comes with some limitations. As certificate handling is the browsers responsibility, handling errors such as wrongly chosen client certificates is impossible. When using a reverse proxy, this problem can be mitigated by returning appropriate error sites.

[2] https://github.com/legastero/stanza.io

[3] https://discourse.igniterealtime.org/t/wrong-field-type-of-pubsub-node-type-and-how-to-update-it/81596

More crucially, errors in the reverse proxy or *XMPP* server configuration, such as missing client certificates, are hard if not impossible to detect on the client. Indicators for a misconfigured proxy can be *HTTP* status codes, which Stanza.io does sadly not expose.

4.3 Code Quality

As our *XMPP-Grid broker* implementation is intended to be a maintainable, production-ready application rather than a prototype, we have placed much emphasis on code quality. The measures taken can broadly be divided into three categories: technical measures, strategic decisions and processes.

Technical Measures and Strategic Decisions

Using Angular and the default Angular CLI was mostly a strategic decision. Deviating as little as possible from the standard configuration ensures long-term maintainability, better security and relatively straight-forward upgrades to newer Angular versions. Another benefit of the Angular CLI project setup is that it comes with codelyzer[4] (including tslint) for static code analysis and style linting.

Apart from using the built-in linting mechanism, we followed Angular's style guide [2]. Using IntelliJ Ultimate[5] turned out to be particularly helpful as they give quick feedback for frequent mistakes and even violations of the angular style guide.

We would have preferred to use more tools, especially for code metrics such as Lack of Cohesion of Methods (LCOM), and Afferent/Efferent Coupling. However, we were not able to find tools that were actively maintained and work with TypeScript.

Processes

On the process side, we tried to apply test driven development as much as possible. Doing so turned out to be harder than expected as Angular's component testing infrastructure deviates from a real web browser environment (see section *4.4 Testing*).

Another process we heavily relied on to improve code quality and security were code reviews. Each change, for the documentation and code, was reviewed using GitHub pull-requests[6]. In most cases, minor changes were detected and addressed during these reviews. Continuous integration with TravisCI[7] ensured that these changes never contained compilation errors or failing tests.

We also regularly discussed architectural and structural questions in our retrospectives and standup meetings.

In general, writing clean, modular and testable code has been our main priority.

4.4 Testing

High quality tests are inevitable for long-lived software projects. They help developers to ensure that everything (still) works as expected after a change. For the

[4] http://codelyzer.com/
[5] https://www.jetbrains.com/
[6] https://www.github.com/
[7] https://travis-ci.com/

XMPP-Grid broker, we focused on unit and end-to-end tests. Following the principles of the test pyramid [9], we wrote many fast and cheap unit tests verifying the fundamental behaviour and fewer complex and expensive end-to-end tests.

Unit Tests

Testing the Angular services was rather straightforward with the aid of Jasmine and its mocking functionality. We deliberately abstained from using Angular's testing framework for services to keep tests simple and comprehensible. Since the primary task of most services is to send and receive *XMPP*-commands, integration and end-to-end tests are better suited in most cases.

Writing tests for Angular components was not always essential, as actual rendering in a web browser is required. To off fine-grained control and to be able to conduct tests, Angular provides a rather complex set of testing tools. Because of this indirection, tests are conceptually not identical with the actual Angular application, making test driven development harder if not impossible.

End-to-End Tests

The end-to-end tests were written using Protractor[8], Angular's official end-to-end testing framework. Protractor starts the development setup and verifies the application using a remote-controlled browser.

End-to-end tests are usually more challenging to write than unit tests, as different types of race conditions and varying delays to backend applications can occur. Protractor usually resolves these issues with the aid of Zone.js, a library that creates "execution context[s] that persists across async tasks" called zones. To create zones, Zone.js intercepts most web browser APIs, like *HTTP* requests. [3]

Because Zone.js is aware of all open *HTTP* requests, Protractor can wait until a request has been completed before continuing with test execution.

However, due to our use of *BOSH* in the end-to-end tests (see section *3.1.3 Implemented Web Application Topology*), we could not benefit from the Zone.js change detection. *BOSH* uses *HTTP* long polling to communicate with the *XMPP* server, which leads to a Zone that always has open requests [27].

Therefore, we had to manually implement waiting conditions.

Writing tests paid off quickly as they promptly caught many potential bugs introduced by small changes and refactorings.

4.5 Documentation

Installation instructions and security best practices are directly documented in the git source code repository using the plain text file format called AsciiDoc. Interested parties can browse the documentation directly on GitHub, which is not uncommon in the open source community.

A compact getting started guide for developers is also available in the source code repository. The source code has JSDoc[9] based documentation optimised for compodoc[10], a "documentation tool for your Angular applications".

As already discussed in section *3.1 Architecture*, all architectural decisions were documented systematically. These decisions enable new developers and interested

[8] http://www.protractortest.org/
[9] http://usejsdoc.org/
[10] https://compodoc.app/

parties to comprehend why certain decisions were made. With the idea of making project documentation durable, all decisions were written in the same plaintext format as the other project documentation.

Chapter 5

Discussion and Conclusion

"Wisdom is not a product of schooling but of the lifelong attempt to acquire it."
— Albert Einstein

5.1 Achieved Result

In this section, we describe the achieved results during this thesis and how we managed to reach them.

5.1.1 Implemented Requirements

As listed in table *5.1*, we implemented about 86% of the overall requirements that we had planned to accomplish. The five remaining requirements could not be implemented due to technical constraints as discussed in depth in section *4.2 Encountered Problems*. To compensate for it, we implemented two optional requirements.

Requirement Group	implemented
A.6.1 Authentication	*partial (4/7)*
A.6.2 List Topics and Collections	*partial (5/6)*
A.6.3 Create a New Topic	*complete (1/1)*
A.6.4 Create a New Collection	*complete (3/3)*
A.6.5 Delete an Existing Topic	*complete (1/1)*
A.6.6 Delete an Existing Collection	*complete (3/3)*
A.6.7 Manage Topic/Collection Subscriptions	*complete (5/5)*
A.6.8 Manage Topic Affiliations	*complete (4/4)*
A.6.9 Manage Persisted Items of a Topic	*partial (4/5)*
A.6.10 Manage Subscription Requests (optional)	*not implemented*
A.6.11 Validate Controller Configuration (optional)	*complete (2/2)*
Total	$32/37 \approx 86\%$

TABLE 5.1: Fulfilled requirements by groups.

5.1.2 Architecture

Concurrency, Scalability and Performance

Due to our chosen architecture style (see section *3.1 Architecture*), concurrency, scalability and performance are primarily the concern of the *XMPP* server.

Our implementation submits queries to the *XMPP* server in parallel whenever possible and reduces redundant queries via data sharing.

Usability

Usability was a priority in our application and we implemented several features for ease of use. A good example is the use of so-called bread-crumbs, which allow fast and direct navigation through different application levels.

We regret that it was not possible to conduct a usability test with a typical user during the thesis.

Security

In an expert review of our architecture, a high level of security was attested.

To prevent risks due to misconfiguration or missing features of the *XMPP* server or reverse proxy, we added additional documentation alongside the application, containing recommendations for administrators. More details on this can be found in section *3.4 Security Risk Mitigation* and the docs folder in the source code repository.

Architectural Decisions

In this section, we reflect on our *Architectural Decisions* and how they turned out.

Architecture Style Due to limitations of the *XEPs*, features like autocomplete and filtering could not be implemented. This would probably have worked better with a server plug-in, but would have resulted in close coupling to a specific *XMPP* server.

Platform The implementation of a web application proved portable and flexible as intended.

SASL Authentication Strategy The use of *SASL EXTERNAL* proved to be suboptimal. We discovered that due to the chosen architecture and policies in current web browsers, a reverse-proxy is nearly always required (see section *4.2.1 Multiple Administrators*).

In hindsight, to use *SASL SCRAM* with username and password would probably have eased the development and deployment of the application.

Role Management We are convinced that the decision to model role management with collection nodes is an ideal solution. However, we were not able to verify this functionality, as Openfire has not implemented collection nodes according to the latest version of the *publish-subscribe XEP* draft [34].

Web Application Communication Topology In general, using *XMPP* directly from the web browser worked well. However, due to the incomplete WebSocket implementation in Openfire and the browsers policies concerning *SASL EXTERNAL*, we had to use *BOSH* and an *HTTP* proxy in front of the *XMPP* server. See section *4.2 Encountered Problems* for more details.

Frontend Framework The decision to use Angular with TypeScript in combination with the IntelliJ IDEA IDE has turned out to be an efficient and clean solution.

UI Library The decision for the Spectre.css[1] library provided us with a reasonable compromise regarding productivity and long-term maintainability.

Frontend Structure To split the application into multiple modules worked well and helped to structure the code. We had to slightly modify the initial design in the course of the project, to address the increasing complexity.

XMPP Client Library The Stanza.io *XMPP* library has served its purpose. We opened two pull requests with error corrections on GitHub[2], which were quickly merged and released.

5.1.3 Implementation

Tests

As described in section *4.4 Testing*, good tests and a solid test coverage are important for a long-lived project.

To measure unit tests coverage, we used the istanbul coverage tool[3]. We achieved a total of 93.69% statement coverage, thanks to our comprehensive set of unit tests.

The code coverage achieved using the integration tests is not included in the test coverage, as no such tooling exists.

In total, we have approximately 2.25 test code lines per line of application code.

Category	**Lines of Code**
Typescript Application Code	2'060
HTML Application Code	600
CSS Application Code	163
Total Application Code	**2'823**
Unit Test Code	5'347
Integration Test Code	994
Total Test Code	**6'341**

TABLE 5.2: Lines of code by category excluding third-party code.

[1] https://picturepan2.github.io/spectre/

[2] See https://github.com/otalk/jxt-xmpp/pull/23 and https://github.com/legastero/stanza.io/pull/264

[3] https://gotwarlost.github.io/istanbul/

Test Category	Number of Tests
Unit Tests	305
Integration Tests	19
Total	**324**

TABLE 5.3: Number of tests per test category.

5.2 Lessons Learned

In this section, we describe unexpected project events and the lessons we learned from them.

5.2.1 Project Course

Issues and Time Management

In general, our issue management and time tracking with JIRA[4] and our Scrum-based approach worked very well.

While discussing time management issues in retrospective 3, we noted that we significantly underestimated the required time for several implementation issues. Many implementation issues were quite comprehensive, in some cases estimated at more than hours.

To address these estimation issues, we decided to create smaller issues and list tangible subtasks in the form of check-lists. A check-list extension for JIRA facilitated this task.

Despite the reduced task sizes, estimating and specifying tasks precisely remained a challenge. Our limited experience with the Angular framework and the *XMPP* ecosystem were undoubtedly large contributing factors.

Documentation

To accomplish high-quality documentation, we used GitHub pull requests to carry out peer reviews. To simplify this process, we also set up continuous integration builds which always posted the latest stable documentation and appendices on the project website. We think that this approach led to a high overall documentation standard.

It was difficult to summarise the technical background and describe our architecture due to the different terminology used by the *XMPP* and IETF standards. We discuss this in section *5.2.4 Standards*.

5.2.2 Architectural Decisions

Architecture-relevant decisions were carried out and justified in the form of architectural design decisions [19] (see appendix *A.3 Architectural Decisions*).

This approach helped us to systematically document influences and plan the architecture in a structured way.

Retrospectively, we should have made more architectural decisions later on in the project, e.g. concerning barrel imports or to establish layering guidelines.

[4] https://www.atlassian.com/software/jira

5.2.3 Development, Frameworks and Tooling

Test Driven Development was not possible in the way we had anticipated.

Due to the use of Angular, the testing environment differed substantially from the actual application context. Therefore, it was challenging to create tests before implementing most of the actual code structure. A factor that also contributed to this difficulty was our prior lack of Angular expertise.

Nevertheless, writing many tests proved to be very valuable. It helped us to be confident during development and will be useful to future developers extending the application.

The Docker Development Environment has proven to be valuable. It provides developers with a very productive way to test modifications in a realistic yet portable environment.

Compodoc, the tool we used to document and visualise the structure of our Angular application, did not add as much value to the project documentation as we had hoped.

We assume that Compodoc is better suited for Angular libraries than applications.

5.2.4 Standards

During the course of this thesis, we learned valuable lessons about working with standards and about the way these standards pose challenges or support development.

Terminology

The *XMPP-Grid standard* uses *SACM* terminology [5], whereas the *XMPP* standard and all *XEPs* use a different terminology. Most concepts and term definitions differ or overlap slightly, making it difficult to comprehend and connect both formats. It also makes the use of a consistent terminology impossible, as some concepts from *XMPP*/*XEPs* are not reflected in *SACM* terminology and vice versa.

XEPs Draft Versions

Many of the used *XMPP* Extension Procotols (*XEPs*) are not yet final but still in the draft phase. Most notably, these include the Publish-Subscribe (XEP-0060 and XEP-0248), Result Set Management (XEP-0059) and *BOSH* (XEP-0206) *XEPs*. Only the core *XEPs*, such as Service Discovery (XEP-0030) and Data Forms (XEP-0004), are declared final.

Because many drafts have not received major updates (XEP-0059, for example, has not been modified for over 10 years) these drafts are treated as de facto standards in the community, neglecting the possibility of significant changes. Unfortunately, not all drafts are stable.

A prominent example of a modified standard draft is the *publish-subscribe XEP*. In the last few years, significant changes have been made and the concept of "Collection Nodes" was even extracted into a separate standard draft [34]. In our case, the Openfire *XMPP* server implemented an older version of this *XEP*, not supporting all features that we planned to use in this thesis.

Deprecated *XEPs*

Many *XEPs* build on functionality specified by other, cross-referenced *XEPs*. This is problematic, especially as some referenced standards are not active anymore.

An example is the PubSub Collection Nodes *XEP* [34], which currently has a *deferred* status, but is still used in the latest version of the *publish-subscribe XEP* [23], which currently has a *draft* status.

Non-Binding Standardisation

Many features required for the *XMPP-Grid broker* implementation are marked as optional in the corresponding *XEPs*. To some extent, the availability of these features can be queried using the feature discovery mechanism [18], but not all optional features are exposed in this way.

The *publish-subscribe XEP* contains multiple such optional features.

Additionally, some features are not explicitly specified in the according *XEP*, but rather implicitly demonstrated using examples.

These limitations make it difficult to rely on the availability of some features described in these *XEPs*.

5.3 Future work

The result of our bachelor thesis is a fully functional application, ready to prove itself in production. Even though all specified functionality was implemented, the user experience can still be further improved.

Conducting usability tests by observing administrators who manage *XMPP-Grids* can reveal significant insights [20].

To further improve the user experience, auto-complete for users and topics might be helpful. As already discussed in section *4.2 Encountered Problems*, this cannot be implemented efficiently due to shortcomings in the *publish-subscribe XEP*. One option would be to propose the required functionality in the *XEP* standardisation process. A more short-term solution would be to tie the implementation closer to a specific *XMPP* server that supports these features over proprietary APIs. Alternatively, an unofficial *XEP* including corresponding server plugins can be specified and implemented.

Working around shortcomings of the *XMPP* server implementations, lost updates for example (see section *4.2.5 Lost Updates*), could advance the usability as well. However, it must be noted that adding more logic in the client contradicts the *XMPP* philosophy that encourages simple clients and complex server implementations [36].

5.4 Conclusion

The XMPP-Grid broker application enables administrators to configure *XMPP-Grids* in a straight-forward and productive way. The modern web interface facilitates obtaining a comprehensive view of the configuration and structure of an *XMPP-Grid*. Apart from improving the usability significantly, the application is also cross-platform and not tied to a particular *XMPP* server implementation.

Our proposed architecture has proven to work in practice. Although the initial setup with a proxy server is complex, the architecture will pay off in practice regarding security and maintainability as reverse proxies are commonly used, and static sites are easy to maintain and upgrade.

Angular and Stanza.io turned out to be a good choice for the implementation. Angular provides productive tools and a comprehensive testing infrastructure that allowed us to build an application that can be maintained efficiently in the long-term.

Stanza.io met most of our requirements concerning *XMPP* support and allowed us to extend and improve it where needed.

Using Openfire as an *XMPP* server backend was demanding at times due to the scanty implementation of the *publish-subscribe* standard. Some of these shortcomings, however, revealed problematic limitations of the standard which otherwise might not have been considered.

The Bachelor Thesis went well from our point of view. Not only were we able to reach all major requirements, but also deliver a robust and ready-to-use solution.

In the future, the application must prove itself in practice. Based on feedback from users in industry, the usability and feature set can further be refined.

To support some features efficiently, the implementation must either be bound to a specific *XMPP* server or new extensions to the *XMPP* standard must be proposed.

We hope that with the help of our implementation the IETF draft "Using *XMPP* for Security Information Exchange" will become an established security standard used in practical industry applications.

Bibliography

[1] Angular guide: Security. https://angular.io/guide/security, 2017.

[2] Angular style guide. https://angular.io/guide/styleguide#style-guide, 2018.

[3] Zone.js readme. https://angular.io/guide/security, 2018.

[4] Agile Alliance. What are user stories? https://www.agilealliance.org/glossary/user-stories, 2015.

[5] H. Birkholz, J. Lu, J. Strassner, N. Cam-Winget, and A. W. Montville. Security Automation and Continuous Monitoring (SACM) Terminology. Internet-Draft draft-ietf-sacm-terminology-14, Internet Engineering Task Force, Dec. 2017. Work in Progress.

[6] S. O. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119, Mar. 1997.

[7] S. Brown. The c4 model for software architecture. https://c4model.com/.

[8] N. Cam-Winget, S. Appala, S. Pope, and P. Saint-Andre. Using XMPP for Security Information Exchange. Internet-Draft draft-ietf-mile-xmpp-grid-05, Internet Engineering Task Force, Feb. 2018. Work in Progress.

[9] M. Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 1st edition, 2009.

[10] R. Danyliw. The Incident Object Description Exchange Format Version 2. RFC 7970, Nov. 2016.

[11] R. Eatmon, J. Hildebrand, J. Miller, T. Muldowney, and P. Saint-Andre. Data Forms. XEP-0004, Aug. 2007.

[12] C. P. et al. Secure websocket with client certificate not working. Chromium Bug Tracker, 2013. https://bugs.chromium.org/p/chromium/issues/detail?id=329884#c24.

[13] G. I. et al. Manage client certificates on chrome devices. Chrome Documentation, 2013. https://support.google.com/chrome/a/answer/6080885?hl=en#manage-certs.

[14] M. et al. Security/server side tls. MozillaWiki, 2018. https://wiki.mozilla.org/index.php?title=Security/Server_Side_TLS&oldid=1191414.

[15] S. Frei, T. Duebendorfer, and B. Plattner. Firefox (in) security update dynamics exposed. *SIGCOMM Comput. Commun. Rev.*, 39(1):16–22, Dec. 2008.

[16] T. Hansen. SCRAM-SHA-256 and SCRAM-SHA-256-PLUS Simple Authentication and Security Layer (SASL) Mechanisms. RFC 7677, Nov. 2015.

[17] S. Hares, J. Strassner, D. Lopez, L. Xia, and H. Birkholz. Interface to Network Security Functions (I2NSF) Terminology. Internet-Draft draft-ietf-i2nsf-terminology-05, Internet Engineering Task Force, Jan. 2018. Work in Progress.

[18] J. Hildebrand, P. Millard, R. Eatmon, and P. Saint-Andre. Service Discovery. XEP-0030, Oct. 2017.

[19] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, pages 109–120. IEEE, 2005.

[20] S. Krug. *Don't Make Me Think: A Common Sense Approach to the Web (2Nd Edition)*. New Riders Publishing, Thousand Oaks, CA, USA, 2005.

[21] A. Melnikov and I. Fette. The WebSocket Protocol. RFC 6455, Dec. 2011.

[22] A. Menon-Sen, A. Melnikov, N. Williams, and C. Newman. Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms. RFC 5802, July 2010.

[23] P. Millard, P. Saint-Andre, and R. Meijer. Publish-Subscribe. XEP-0060, Feb. 2018.

[24] J. Moffitt. *Professional XMPP Programming with JavaScript and jQuery*. Wrox Press Ltd., Birmingham, UK, UK, 2010.

[25] A. Parsovs. Practical issues with tls client certificate authentication. Cryptology ePrint Archive, Report 2013/538, 2013. https://eprint.iacr.org/2013/538.

[26] I. Paterson, P. Saint-Andre, V. Mercier, and J.-L. Seguineau. Result Set Management. XEP-0059, Sept. 2006.

[27] I. Paterson, D. Smith, P. Saint-Andre, J. Moffitt, L. Stout, and W. Tilanus. Bidirectional-streams Over Synchronous HTTP (BOSH). XEP-0124, Nov. 2016.

[28] E. Rescorla and T. Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Aug. 2008.

[29] A. Saini, M. S. Gaur, and V. Laxmi. The darker side of firefox extension. In *Proceedings of the 6th International Conference on Security of Information and Networks*, SIN '13, pages 316–320. ACM, 2013.

[30] P. Saint-Andre. Streaming xml with jabber/xmpp. *IEEE Internet Computing*, 9(5):82–89, Sept 2005.

[31] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120, Mar. 2011.

[32] P. Saint-Andre. Jabber Component Protocol. XEP-0114, Jan. 2012.

[33] P. Saint-Andre and D. Cridland. XMPP Extension Protocols. XEP-0001, Nov. 2016.

[34] P. Saint-Andre, R. Meijer, and B. Cully. PubSub Collection Nodes. XEP-0248, Sept. 2010.

[35] P. Saint-Andre and me@thijsalkema.de. Use of Transport Layer Security (TLS) in the Extensible Messaging and Presence Protocol (XMPP). RFC 7590, June 2015.

[36] P. Saint-Andre, K. Smith, and R. Tronon. *XMPP: The Definitive Guide Building Real-Time Applications with Jabber Technologies*. O'Reilly Media, Inc., 2009.

[37] A. Steffen. XMPP-Grid HSR Rapid-Prototype. https://github.com/sacmwg/vulnerability-scenario/blob/b3bf6a5b21f242b788488ba8991595172fe663e7/ietf_hackathon/strongSwan/pubsub_client.py, Nov 2017.

[38] L. Stout, J. Moffitt, and E. Cestari. An Extensible Messaging and Presence Protocol (XMPP) Subprotocol for WebSocket. RFC 7395, Oct. 2014.

[39] A. van Kesteren. Cross-origin resource sharing. W3C recommendation, W3C, Jan. 2014. http://www.w3.org/TR/2014/REC-cors-20140116/.

[40] D. Veditz, M. West, and A. Barth. Content security policy level 2. W3C recommendation, W3C, Dec. 2016. https://www.w3.org/TR/2016/REC-CSP2-20161215/.

[41] R. Wirdemann. *Scrum mit User Stories*. Carl Hanser Verlag München, 2017.

[42] K. Zeilenga and A. Melnikov. Simple Authentication and Security Layer (SASL). RFC 4422, June 2006.

List of Figures

2.1 XMPP example overview 4
2.2 DSL of the XMPP-Grid standard 6
2.3 DSL of used XMPP XEPs 6

3.1 Architecture context diagram 7
3.2 Architecture container diagram: Rich client 8
3.3 Architecture container diagram: Web application 9
3.4 Architecture container diagram: Web proxy 9
3.5 Architecture container diagram: Web application with proxy 10

4.1 Development setup deployment diagram 16

1 Use case diagram LXVI
2 Login-screen wireframe LXXIII
3 Controller overview wireframe LXXIII
4 All collections wireframe LXXIV
5 All topics wireframe LXXIV
6 New collection wireframe LXXIV
7 New topic wireframe LXXV
8 Collection overview wireframe LXXV
9 Topic overview wireframe LXXV
10 Topic/Collection affiliations wireframe LXXVI
11 Topic/Collection configuration wireframe LXXVI
12 Topic parent collections items wireframe LXXVI
13 Persisted items wireframe LXXVII

List of Tables

5.1 Fulfilled requirements by groups. 23
5.2 Lines of code by category excluding third-party code. 25
5.3 Number of tests per test category. 26

Glossary

BOSH

Bidirectional-streams Over Synchronous *HTTP* [27]

Broker

See *XMPP-Grid broker*

Component

An encapsulation of software that communicates using Interfaces, that is composed of *SACM* capabilities. [5, 17]

Consumer

"In *SACM*, an entity that contains functions to receive information from other *components*; as used here, the term refers to an *XMPP publish-subscribe* Subscriber." [8]

Control Plane

"An architectural component that provides common control functions to all *SACM* components." [5]

Controller

"In SACM, a 'component containing control plane functions that manage and facilitate information sharing or execute on security functions'; as used here, the term refers to an *XMPP* server, which provides core message delivery [RFC6120] used by publish-subscribe entities." [8, 5]

Data Forms

XMPP Data Forms Extension

HTTP

Hypertext Transfer Protocol

IETF

Abbreviation for Internet Engineering Task Force

Info/Query

XMPP info/query *stanza*

Jabber

Originial Name of *XMPP*

Jabber Search

XMPP/Jabber Search Extension, historically used to search for jabber users, eg. in the "Jabber User Directory"

JID

Jabber IDentifier, e.g. bob@example.com/mobile

Message

XMPP message *stanza*

MILE

Abbreviation for the 'Managed Incident Lightweight Exchange' working group

Persisted Item

XMPP messages that are persisted in a *topic*.

Platform

"Any entity that connects to the *XMPP*-Grid in order to publish or consume security-related data." [8]

Presence

XMPP presence *stanza*

Provider

"In *SACM* An entity that contains functions to provide information to other *components*; as used here, the term refers to an *XMPP publish-subscribe* Publisher." [8]

Publish-Subscribe

XMPP Publish-Subscribe Extension

PubSub

Common abbreviation for *publish-subscribe*

Result Set Management

XMPP Result set Management Extension

SACM

Abbreviation for Security Automation and Continuous Monitoring

SASL

"The Simple Authentication and Security Layer (SASL) is a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms." [42]

SASL EXTERNAL

A *SASL* authentication mechanism, which is not directly implemented by *SASL* but rather by external means to authenticate the client, e.g. *TLS*. [42]

SASL SCRAM

"a family of [SASL] authentication mechanisms called the Salted Challenge Response Authentication Mechanism (SCRAM)" [22]

Service Discovery

XMPP Service Discovery Extension

Stanza

An *XMPP* a fragment of XML that is sent over a stream, e.g. a *message*.

TCP

Transmission Control Protocol

TLS

Transport Layer Security [28]

Topic

"A contextual information channel created on a Broker at which messages generated by a Provider are propagated in real time to one or more Consumers. Each Topic is limited to a specific type and format of security data (e.g., IODEF) and provides an *XMPP* interface by which the data can be obtained." [8]

XEP

XMPP Extension Protocol

XML

eXtensible Markup Language

XMPP

eXtensible Messaging and Presence Protocol

XMPP-Grid

"A method for using the Extensible Messaging and Presence Protocol (*XMPP*) [RFC6120] to collect and distribute security-relevant information among network *platforms*, endpoints, and any other network-connected device." [8]

XMPP-Grid broker

"A *SACM* broker Controller is a *controller* that contains *control plane* functions to provide and/or connect services on behalf of other *SACM* components via interfaces on the *control plane*" [5]

XMPP-Grid standard

The *IETF MILE* 'Using XMPP for Security Information Exchange draft-ietf-mile-xmpp-grid-05' draft. [8]

Appendices

A.1 Project Plan

XMPP-Grid Broker: Project Plan

Authors:
Fabian HAUSER and
Raphael ZIMMERMANN

Advisor:
Prof. Dr. Andreas STEFFEN

Spring Term 2018

Contents

Contents **i**

1 Project Overview **1**

2 Project Organisation **2**
2.1 Roles . . . 2

3 Project Management **3**
3.1 Components . . . 3
3.2 Time Budget . . . 3
3.3 Schedule . . . 3
3.3.1 Iterations & Milestones . . . 3
3.3.2 Meetings . . . 4

4 Risk Management **6**

5 Infrastructure **8**
5.1 Project Management and Development . . . 8
5.1.1 Development Tools . . . 8
5.2 Backup and Data Safety . . . 8

6 Quality Measures **9**
6.1 Documentation . . . 9
6.2 Project Management . . . 9
6.2.1 Sprint Planning . . . 9
6.2.2 Definition of Done . . . 9
6.3 Development . . . 10
6.4 Testing . . . 10

Bibliography **I**

List of Figures **II**

List of Tables **III**

Chapter 1

Project Overview

The goal of the bachelor thesis is to build a broker application and graphical user interface to administer XMPP-Grids according to draft-ietf-mile-xmpp-grid, as described in the task description [3].

Chapter 2

Project Organisation

All team members have the same strategic rights and duties. Prof. Dr. Andreas Steffen is our project advisor.

2.1 Roles

Due to the small team size, most roles are performed by both team members.

Raphael Zimmermann
: project management, software engineering, quality assurance.

Fabian Hauser
: infrastructure management, software engineering, quality assurance.

Chapter 3

Project Management

3.1 Components

For a better overview and to allow us a sophisticated time assessment, we decided to group tasks into categories, i.e. JIRA components. Components represent processes, documents and products which are to be released.

Currently, tasks are separated into following components:

- Application
- Final Submission Document
- Management
- Poster
- Presentation
- Project Plan

3.2 Time Budget

The project started with the Kickoff Meeting on 19.02.2018 and will be completed after 16 weeks by 15.06.2018. The two team members are available for 360 hours each during the semester which corresponds to a weekly time budget of 20 hours per person and two weeks with a weekly time budget of 40 hours per person.

Apart from the statutory holidays, there are no further absences planned.

3.3 Schedule

The project schedule is an iterative process based on elements of SCRUM.

We decided on a sprint duration of approximately one week, but allow deviations in working hours depending on statutory holidays.

3.3.1 Iterations & Milestones

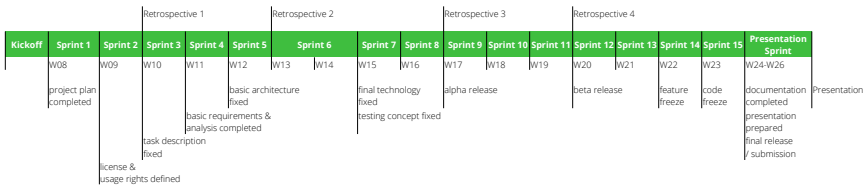


FIGURE 3.1: Overview of the iterations

The goals and milestones resulting from each sprint are shown in Figure 3.2.

TABLE 3.1: Meeting Time Budget

Meeting Type	Total Duration per Person	Total Duration for the Team
Supervision Meetings	12 hours	24 hours
Standup Meetings	16 hours	32 hours
Sprint Planning Meetings	18 hours	36 hours
Retrospective	4 hours	8 hours
Total	50 hours	100 hours

3.3.2 Meetings

The team works every Monday (08:00 - 12:00) and Tuesday (08:00 - 17:00) together in the study room and remotely Fridays (08:00 - 17:00). Wednesday and Friday begin with a daily stand-up meeting taking no longer than 15 minutes. Sprint planning meetings are carried out on Tuesday at 10:00. Table 3.1 shows an overview of the total meeting time budget.

Regular meetings with the project advisor usually take place on Monday in Prof. Dr. Steffen's office.

Raphael Zimmermann will take meeting notes for every meeting. Meeting minutes are published as appendix of the thesis afterwards.

Sprint	**Sprint 1**	**Sprint 2**	**Sprint 3**	**Sprint 4**
Tag	**0.1.0**	**0.2.0**	**0.3.0**	**0.4.0**
Date	20.02.2018 – 27.02.2018	27.02.2018 – 06.03.2018	06.03.2018 – 13.03.2018	13.03.2018 – 20.03.2018
Milestones	- license & usage rights defined	- task description fixed	- basic requirements & analysis completed	- basic architecture fixed
Time Budget	40	40	40	40
Tasks	- setup remaining project infrastructure - read ietf draft, XEP standards etc. - compile list of open questions - begin with chapter "initial situation" - analyze existing python proof-of-concept	- collect non-functional requirements (NFRs) - compile set of user stories - draw wireframes - research frameworks and technology. Prepare architectural decisions	- extend NFRs, user stories, wireframes - make first architectural decisions - further framework and technology research - implement proof of concepts for ciritical components	- make big architectural decisions - implement proof of concepts for ciritical components
Documents / Chapters / Artefacts		- chapter "Initial Situation" completed - signed task description		

Sprint	**Sprint 5**	**Sprint 6**	**Sprint 7**	**Sprint 8**
Tag	**0.5.0**	**0.6.0**	**0.7.0**	**0.8.0**
Date	20.03.2018 – 27.03.2018	27.03.2018 – 10.04.2018	10.04.2018 – 19.04.2018	19.04.2018 – 24.04.2018
Milestones		- final technology fixed - testing concept fixed		
Time Budget	40	80	40	40
Tasks	- implement proof of concepts for ciritical components - draft testing concept	- implement proof of concepts for ciritical components - finalise testing concept	- implement most important user stories - make further architectural decisions	- implement most important user stories - make further architectural decisions - write integration tests
Documents / Artefacts		- Chapter "Our Approach" completed		

Sprint	**Sprint 9**	**Sprint 10**	**Sprint 11**	**Sprint 12**
Tag	**0.9.0**	**0.10.0**	**0.11.0**	**0.12.0**
Date	24.04.2018 – 01.05.2018	01.05.2018 – 08.05.2018	08.05.2018 – 15.05.2018	15.05.2018 – 22.05.2018
Milestones	- alpha release (29.04.2018)			
Time Budget	40	40	40	40
Tasks	- Implement secondary user stories - write integration tests	- Implement secondary user stories - write integration tests	- Implement secondary user stories - write integration tests	- Implement remainging user stories - improve code base - write integration tests
Documents / Artefacts	- alpha release bundle			

Sprint	**Sprint 13**	**Sprint 14**	**Sprint 15**	**Presentation Sprint**
Tag	**0.13.0**	**0.14.0**	**1.0.0**	**-**
Date	22.05.2018 – 31.05.2018	31.05.2018 – 06.06.2018	06.06.2018 – 13.06.2018	13.06.2018 – 27.06.2018
Milestones	- beta release - future freeze	- code freeze	- documentation completed - final release / submission	- presentation prepared - presentation
Time Budget	40	50	80	30
Tasks	- Implement remainging user stories - improve code base - write integration tests	- Improve code base - write integration tests - write documentation	- fix eventual bugs - write documentation - submit documents	
Documents / Artefacts	- beta release bundle	- abstract - management summary	- final release bundle - poster - personal reports - time accounting	- presentation

FIGURE 3.2: Detailed overview with tasks and milestones of all sprints.

Chapter 4

Risk Management

An assessment of the project-specific risks is carried out in Table 4.2 as time loss during the whole project. The risk matrix in Table 4.1 provides an overview of the risk weighting.

To account for these risks, we reduce our weekly sprint time by the total weighted risk applicable to the planned task topics (on average approximately 13.5%). We also review the risk assessment after every sprint, adapt it and take measures if necessary.

Probability \ Severity	High ($\geq 5d$)	Medium (2-5d)	Low ($\leq 2d$)
High ($\geq 60\%$)	1		
Medium (30-60%)	6		
Low ($\leq 30\%$)		3, 4, 5	

TABLE 4.1: The risk matrix. Numbers reference to the risk assessment Table 4.2

TABLE 4.2: Risk assessment table. Time in hours over the total project duration.

#	Title	Description	Prevention / Reaction	Risk [h]	Probability	= [h]
1	Incomplete reference documentation	The reference documentation / standards are incomplete or difficult to comprehend.	Discuss missing parts with project advisor	60	60%	36
2	Communication errors	Errors due to miscommunication or misapprehension.	Maintain a high level of interaction, precise specification of tasks responsibilities, conduct meetings if ambiguities exist.	30	50%	15
3	Problems with project infrastructure	The used project infrastructure is not or only partially available, or data loss occurs within management software.	Clean setup and self-hosting of the tools to prevent third-party dependencies.	45	30%	13.5
4	Scope creep	The project's scope is extended over the project course.	Define the project scope and limitations precisely. Discuss changes with the project advisor.	45	30%	13.5
5	Dependency errors	There are errors/bugs in third-party dependencies, i.e. libraries.	Carefully select libraries and limit thid-party dependency to a minimum.	30	30%	9
6	Missing dependency documentation	Selected libraries are lacking proper documentation	The documentation quality of a library should be a selection criterion.	30	40%	12
	Total weighted risk					99

Chapter 5

Infrastructure

5.1 Project Management and Development

For project management we utilise JIRA [1], hosted on our HSR project server, which runs a standard Ubuntu Linux 17.10.

As document/code storage, git repositories on GitHub are used, the continuous integration/deployment will be defined in the course of the project.

5.1.1 Development Tools

The development tools will be defined in the course of the project.

5.2 Backup and Data Safety

An incremental backup of the project server including the source code and documentation is created on an independent system every night.

As our documents and code is stored in a git repository, they are also distributed on all development systems.

Chapter 6

Quality Measures

To maintain a high standard of quality, we take the following measures:

- short sprint reviews
- four extended retrospectives
- code reviews
- automated unit and integration testing
- publish all documentation on the project website using continuous integration/delivery.
- using continuous integration for source code

6.1 Documentation

The official documents such as the final submission document, meeting minutes as well as this project plan are written in LaTeX respectively ASCIIDoc and published on the project website[1] containing all PDF documents.

The sources are in both cases kept under version control in the same repository, which allows us to use the same tools and processes for documentation and code. The continuous integration server builds and publishes the website whenever new changes are pushed to the repository.

6.2 Project Management

Because the project plan allows for an iterative process, we use JIRA with its SCRUM-features (such as sprint creation or boards) for project management.

6.2.1 Sprint Planning

Each sprint is mapped to JIRA, which allows the project advisor to trace the project progress. Sprints are represented as boards on which the current state and assignee of any issue is easily visible ("To Do", "In Progress", "Review", "Done").

6.2.2 Definition of Done

An issue may be closed if ***all*** of the following conditions are met:

- All functionality conforms to the specification. Any deviations must be discussed and decided by the team.
- The source code is reasonably documented.

[1] https://xgb.redbackup.org

- No code is commented out.
- No warnings and errors by the compiler or any other quality tool.
- A review is performed and accepted in a pull request.
- The corresponding branch is merged into the stable branch (e.g. master).
- All documents are up to date including the project website.
- Reasonable unit and integration tests exist and pass.
- The complete continuous integration pipeline works.
- All time is logged.

6.3 Development

We decided to use GitHub Flow[2], a straightforward development workflow.

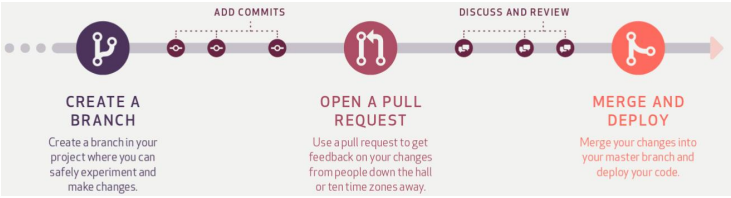


FIGURE 6.1: GitHub Flow illustrated (Source [2])

Since the effective technology will be fixed later in the project, concrete coding guidelines, tools, metrics and an error policy will be defined when appropriate.

6.4 Testing

All functionality must be automatically testable using continuous integration. Any non-trivial function/method must be verified with unit tests.

Integration tests verify extended test scenarios.

Bibliography

[1] Atlassian Inc. Open Source Services by Atlassian Inc. https://developer.atlassian.com/opensource/, 2018.

[2] Github Inc. Github Flow, 2018.

[3] A. Steffen. Task Description "xmpp-grid broker". https://xmpp-grid-broker.github.io/documents/final-submission-document.pdf, 2018.

List of Figures

3.1 Overview of the iterations 3
3.2 Detailed overview with tasks and milestones of all sprints. 5

6.1 Organigram 10

List of Tables

3.1 Meeting Time Budget 4

4.1 Risk matrix 6
4.2 Risk assessment 7

A.2 Development Guide

1. Tools

- Git >= 2.0 for version control

2. Writing Guidelines

In order to have a consistent style of writing, we defined the following guidelines. These guidelines apply to all documents related to the redbackup project.

- Keep it brief, clear and objective
- Write short and straightforward sentences
- Do not use synonyms for concepts etc. (always use the same wording, e.g. 'client' or 'node')
- Abbreviations must be introduced the first time they occur in the text (except well-known ones)
- Prevent ambiguity in sentences
- Use personal style ("we") whenever appropriate; usually for the description of our work.
- When a gender-specific pronoun is required, use "he/she".
- Use present tense except for the description of (our) completed work.

3. Definition of Done

To maintain our high quality needs, we determined following definition of done guidelines:

- All functionality conforms to the specification. Any deviations must be discussed and decided by the team.
- A review is performed and accepted in a pull request.
    - The source code is reasonably documented.
    - No code is commented out.
    - No warnings and errors by the compiler or any other quality tool.
    - Reasonable unit and integration tests exist and pass.
    - All documentations are up to date including the project website.
    - The complete continuous integration pipeline works.
    - The code is formatted according to the guidlines (i.e. according to RustFmt)
- The corresponding branch is merged into the stable branch (e.g. master).
- All time is logged.

A.3 Architectural Decisions

1. Architecture Style

There are three common variants to participate in XMPP communication and manage server configurations: As XMPP server plugin, XMPP component or an XMPP Client/Bot.

The implementation style fundamentally restricts the set of implementation languages and has a profound impact on the fundamental architecture.

The following aspects must be taken into account to find the most suitable architecture style:

- All required management functionality must be supported over the available APIs and protocols.
- Compatibility with most XMPP servers
- Keep the implementation complexity as low as possible

1.1. Considered Options

- XMPP server plugin, e.g. for Openfire Plugin.
- XMPP component (XEP-0114).
- XMPP Client/Bot

1.2. Decision Outcome

Chosen option: XMPP Client/bot, because it is not coupled to a specific XMPP server as the Server Plugin and, in contrast to the XMPP component, supports strong authentication.

1.3. Pros and Cons of the Options

1.3.1. Server Plugin

- Good, because all features could be implemented directly on the XMPP server.
- Good, because there is minimal protocol overhead and abstraction.
- Bad, because a very high coupling to a specific XMPP server is required and compatibility/interoperability is therefore limited
- Bad, because the high coupling to a specific XMPP server usually limits the possible implementation language.

1.3.2. XMPP Component

- Good, because the application style fits very well in the components model.
- Bad, because the specification of components is marked as *Historical* and might therefore not be implemented by many XMPP Servers.
  - Note: Openfire supports components
- Bad, because some XMPP client libraries might support components.
- Bad, because the authentication mechanisms might not suffice the required standards of the XMPP-Grid standard.
- Bad, because it uses a own handshake based digest authentication message.

1.3.3. XMPP Client/Bot

- Good, because a Bot is basically a normal XMPP client, which is supported by every XMPP server
- Good, because all XMPP client libraries implement this feature.
- Good, because secure connections to the XMPP server are supported (SASL).
- Bad, because the application is conceptually not a normal XMPP-Client.

2. Platform

The chosen platform has a fundamental impact on the resulting application as well as on the user experience: Where is it used, from whom and from which device.

2.1. Considered Options

- Client Application with Command Line Interface
- Client Application with Graphical User Interface
- Web application

2.2. Decision Outcome

Chosen option: Web Application because it can easily be installed, updated and provides maximal user acceptance. The development team has experience in writing web applications meaning they know common pitfalls and limitations of the platform. Command line users might use the already existing proof of concept instead.

2.3. Pros and Cons of the Options

2.3.1. Client Application with Command Line Interface

- Good, because cross-platform support can be achieved relatively easy (depending on the chosen language/runtime).
- Good, because the development team has experience writing command line applications.
- Good, because all OS-functionality is available.
- Good, because command line tools are relatively simple to implement.
- Bad, because additional runtimes might be required (python, JVM, .NET etc.).
- Bad, because not all users appreciate command line tools.
- Bad, because cross-platform testing is expensive.
- Bad, because mobile devices (phones, tablets, chromebooks) are not supported.
- Bad, because updates must be performed on every client.
- Bad, because additional rights are required for setup.

2.3.2. Client Application with Graphical User Interface

- Good, because graphical interfaces have a broad acceptance.
- Good, because all OS-functionality is available.
- Bad, because additional runtimes might be required (python, JVM, .NET etc.).
- Bad, because additional libraries are required (e.g. QT).
- Bad, because cross-platform testing is expensive.
- Bad, because mobile-devices (phones, tablets, chromebooks) are not supported.
- Bad, because updates must be performed on every client.
- Bad, because additional rights are required for setup.

- Bad, because the development team has no experience writing graphical user interfaces.

2.3.3. Web Application

- Good, because graphical interfaces have a broad acceptance.
- Good, because the development team has experience writing command line applications.
- Good, because no additional runtime on the client side (except the browser) is required.
- Good, because no additional rights are required for setup.
- Good, because updates must only be performed on the server side (clients need no updates but the platform, i.e. the browser).
- Good, because it scales well.
- Bad, because only limited OS-functionality is available.
- Bad, because testing of multiple browsers and browser versions is expensive.
- Bad, because the supported features vary significantly depending on the browser and version.
- Bad, because a backend server is required although its importance might vary on the chosen architecture.
- Bad, because it has a potentially large attack vector depending on how and where it is deployed.

3. SASL Authentication Strategy

User Story: 1.2 Authentication: Secure XMPP Connection

The XMPP-Grid standard states in section 8.3.1 that

> The XMPP-Grid controller MUST authenticate the XMPP-Grid platform either using the SASL EXTERNAL mechanism or using the SASL SCRAM mechanism (with the SCRAM-SHA-256-PLUS variant being preferred over the SCRAM-SHA-256 variant and SHA-256 variants [RFC7677] being preferred over SHA-1 varients [RFC5802]).

The chosen authentication mechanism has an impact on the implementation of the broker application as well as on the XMPP server support.

Note that this architectural decision has no impact on the communication between the controller and other platforms (i.e. other XMPP clients and the XMPP server).

3.1. Considered Options

- SASL SCRAM
  - SASL SCRAM-SHA-256-PLUS is described in RFC 7677.
  - In summary, SASL SCRAM-SHA-256-PLUS means XMPP over TLS with the client authenticating using a username (JID) and password with challenge-response-exchange.
  - In our case, we may assume that SASL SCRAM-SHA-256-PLUS is currently the only variant supporting a strong hash mechanism
- SASL EXTERNAL
  - SASL EXTERNAL is described in RFC 4422 Appendix A.
  - The SASL EXTERNAL mechanism for XMPP in combination with TLS/PKIX certificates is described in XEP-0178.
  - In our case, we may assume SASL EXTERNAL means XMPP over TLS with the client authenticating using an X.509 certificate, as this is currently the only practical available implementation.

3.2. Decision Outcome

Chosen option: SASL EXTERNAL, because it offers the highest security level and is well suited for large-scale deployments.

3.3. Pros and Cons of the Options

3.3.1. SASL SCRAM

- Good, because SASL SCRAM is widely supported and used by XMPP servers.
- Good, because SASL SCRAM is easy to use for small deployments, as it only requires a JID and password for authentication.
- Good, because SASL SCRAM fulfils the XMPP-Grid standard requirements.

- Bad, because the key management is on the XMPP controller, which leads to a single point of failure and might lead to additional administration efforts.
- Bad, because shared keys (i.e. the password) are used, which cannot be limited to a validity date or selectively revoked.
- Bad, because shared keys(i.e. the password) are difficult to manage in a large scale, decentralised and automated infrastructure.

3.3.2. SASL EXTERNAL

- Good, because a very high-security level can be achieved with X.509 certificates.
- Good, because certificates can be integrated efficiently in a large scale, (mostly) decentralised and automated infrastructure.
- Good, because SASL SCRAM fulfils the XMPP-Grid standard requirements.
- Bad, because a Certification Authority (CA) must be used to issue certificates, which may lead to additional administration efforts and complexity, especially for small deployments.
- Bad, because not all XMPP server and clients have comprehensive support for SASL EXTERNAL.
    - Openfire does not support SASL EXTERNAL with WebSockets, however, it is supported with BOSH.
    - Ejabberd only supports SASL EXTERNAL in the paid professional edition.

4. Role Management

The XMPP Publish-Subscribe mechanism (XEP-0060) lacks an explicit description of how to implement role-based authentication for topics(nodes). It implies two possibilities on how to do so: collection nodes and the roster access model.

The following aspects must be taken into account:

- Preferably use existing XMPP protocols/mechanisms
- The chosen strategy must be usable in most practical use-cases.

4.1. Considered Options

- Usage of Collection Nodes (XEP-0248)
- Roster Access Model which supports groups (XEP-0144)
- Custom Roles implemented on the broker

4.2. Decision Outcome

Chosen option: Usage of Collection Nodes (XEP-0248), because the mechanism is the least intrusive that yet allows a powerful role-concept. We must, however, keep in mind that the XEP is deferred and might therefore not be fully supported by all XMPP servers.

4.3. Pros and Cons of the Options

4.3.1. Usage of Collection Nodes (XEP-0248)

- Good, because existing XMPP mechanisms can be used.
- Good, because with it few consumers can subscribe to many topics indirectly.
- Good, because publishing permissions are managed on a topic level which encourages per-device topics.
- Good, because this mechanism is used in production.
- Bad, because XEP-0248 is deferred
- Bad, because the Collection Nodes must be structured in a specific way to support access management

4.3.2. Roster Access Model which supports groups (XEP-0144)

- Good, because existing XMPP mechanisms can be used.
- Good, because XEP-0144 is a draft (not deferred).
- Bad, because the roster concept aims for IM.
- Bad, because the Roster Access Models is discouraged by the IETF XMPP-Grid standard.

4.3.3. Custom Roles implemented on the broker

- Good, because it enables maximal flexibility
- Bad, because existing XMPP mechanisms cannot be used.

- Bad, because the controller must manage its own data instead of delegating it to the XMPP server.
- Bad, because the controller must periodically check that the affiliations are still appropriately configured on the XMPP server.

5. Web Application Communication Topology

As we decided to build a web application, it remains open how the communication the backing XMPP server flows.

5.1. Considered Options

- **XMPP from the Browser with WebSockets (RFC 7395)**
  Directly speak XMPP over a WebSocket connection with the XMPP server.
- **XMPP from the Browser with BOSH (XEP-0206)**
  Directly speak XMPP over an HTTP long polling connection with the XMPP server.
- **XMPP via WebAPI Proxy**
  Create a standalone application that proxies the XMPP server and exposes a web-API to the client (e.g. a RESTful API).

5.2. Decision Outcome

Chosen option: XMPP from the Browser with WebSockets to reduce duplicated code and use standardised XMPP features. In case the required features are not implemented, a fallback to BOSH should be possible.

In comparison to XMPP via WebAPI Proxy, WebSockets simplify the deployment of new features because they must not be added in two places (the UI and the WebAPI). In comparison to BOSH, WebSockets offer a stateful TCP-connection based on relatively modern standards while BOSH support is provided by many frameworks as a fallback option.

5.3. Pros and Cons of the Options

5.3.1. XMPP from the Browser with WebSockets (RFC 7395)

- Good, because when building an SPA-Client, it is sufficient to serve just static files which provides a higher level of security and performance.
- Good, because actively maintained clients exists (e.g. Stanza.io). Some libraries also support fallback to BOSH.
- Good, because existing client-certificates can be re-used.
- Good, because existing XMPP-mechanisms are used.
- Bad, because additional server plugins must be enabled which is an additional attack vector. (Workaround: Implement a custom proxy)
- Bad, because it is not a standard yet (Proposed Standard).
- Bad, because SASL EXTERNAL is not well supported with WebSockets
  - Openfire does not implement SASL EXTERNAL with WebSockets.
  - WebSockets connections over authenticated TLS is not extensively specified, support may therefore vary depending on the browser implementation.
- Bad, because it might lead to security issues (i.e. CSRF) if used in combination with SASL EXTERNAL/TLS, if no strict request-origin verification is done by the server implementation.

5.3.2. XMPP from the Browser with BOSH (XEP-0206)

- Good, because when building an SPA-Client, it is sufficient to serve just static files which provide higher levels of security and performance.
- Good, because actively maintained clients exists (e.g. strophe.js).
- Good, because existing client-certificates can be re-used.
- Good, because existing XMPP-mechanisms are used.
- Bad, because additional server plugins must be enabled which is an additional attack vector. (Workaround: Implement a custom proxy)
- Bad, because it is not a standard yet (Draft).
- Bad, because not all HTTP-features might be implemented by the XMPP server, which might be a security risk.
    - OPTION preflight requests are not supported by Openfire.
    - A reverse HTTP proxy might be used, to support additional HTTP security features.

5.3.3. XMPP via WebAPI Proxy

- Good, because it decouples the client from the effective XMPP calls (separation of concerns).
- Good, because only minimal HTTP and JavaScript features are used leading to a broad compatibility.
- Bad, because the indirection can limit the performance significantly.
- Bad, because a server application must be installed and maintained (security updates).
- Bad, because existing XMPP-mechanisms are not used.

6. Frontend Framework

To create a user interface for the web application frontend, we must utilise a framework. User Interfaces are too complex these days to implement them in pure JavaScript.

The frontend framework fundamentally restricts the set of implementation languages and has a profound impact on the web interface architecture.

The following aspects must be taken into account to find the most suitable architecture style:

- The effort to learn and master the frameworks best practices must be reasonable.
- The framework must be scalable because the web application is not trivial.
- The number of third party libraries must be kept to a minimum to prevent cost-intense maintenance work in the future.

6.1. Considered Options

- SPA with Angular5
- SPA with React
- SPA with Vue.js
- Django App

6.2. Decision Outcome

Chosen option: Angular because it comes with batteries included and we, therefore, must not rely on additional third-party libraries for everyday tasks. Because Angular is opinionated, there is a clear way to do things which will help us to structure the application. For these benefits, we are willing to accept a steeper learning curve.

6.3. Pros and Cons of the Options

6.3.1. SPA mit Angular5

- Good, because it scales well.
- Good, because the performance is automatically optimised for production builds.
- Good, because it comes with all tools required for building and structuring a sophisticated SPA: components, modules, dependency injection and more.
- Good, because it is very popular, especially in (modern) enterprise applications.
- Good, because it has a large community and is built by Google.
- Good, because it is opinionated, there is a clear way to do things.
- Good, because it uses TypeScript that brings additional safety.
- Bad, because it is heavyweight and hard to learn/understand.
- Bad, because API migrations in the past were costly and hard.
- Bad, because it uses TypeScript which brings indirections and cannot be eliminated.
- Neutral: It is a tool for everything and cannot be used partially.

6.3.2. SPA mit React

- Good, because it scales well.
- Good, because it is lightweight.
- Good, because it encourages functional programming and discourages state.
- Good, because it provides components.
- Good, because it uses modern JavaScript features (ES6).
- Good, because it has a large community and is built by Facebook.
- Bad, because its "lightweightness" leads to lots of third-party dependencies.
- Bad, because it is not trivial to learn (e.g. "pseudo" inline HTML requires custom attributes `className`).
- Bad, because it only defines a minimal core and many standard problems must be solved manually.
- Bad, because additional libraries such as Redux must be used and understood to create complex applications.
- Neutral: Uses JS for everything which is a matter of taste.
- Neutral: Can be used partially.

6.3.3. SPA mit Vue.js

- Good, because it scales well.
- Good, because it is lightweight.
- Good, because it provides components.
- Good, because it makes no assumptions about the JavaScript version used.
- Good, because it elegantly separates template (HTML), styling (CSS) and logic (JS).
- Good, because the core team also maintains fundamental additional such as routing and state management.
- Good, because it has a large and active community.
- Bad, because its "lightweightness" leads to lots of third-party dependencies.
- Bad, because it only defines a relatively small core and some standard problems must be solved manually.
- Bad, because there is no standard way for non-trivial setups (complex SPA, automated testing, etc.).
- Bad, because additional libraries such as Vuex must be used and understood to create complex applications.
- Neutral: Can be used partially.

6.3.4. Django App

- Good, because Django has excellent documentation.
- Good, because Django has a (mostly) stable API.
- Good, because Django is used in other projects in the INS.
- Good, because Django has a very active community.
- Bad, because multiple languages (Python and JavaScript) are required in the frontend.

- Bad, complex client logic (e.g. paging) must still be implemented in JavaScript and might require an additional framework.
- Bad, because it is very heavyweight (includes ORM, migrations and much more) with lots of features that we do not need.
- Bad, because it requires a backend instance in comparison to the SPAs.

7. UI Library

To create a modern and aesthetically pleasing cross-browser user experience and to reduce the implementation efforts, a UI-library shall be used.

7.1. Considered Options

- Angular Material: Material Design components for Angular
- Semantic UI Angular 2
- Spectre.css with custom components

7.2. Decision Outcome

Chosen option: Spectre.css with custom components, because it has minimal third-party dependencies.

Positive consequences:

- From a security point of view, a pure CSS library does not require updates.
  - An Angular component library might introduce security vulnerabilities (e.g. XSS).
  - If the CSS library project is discontinued, the XMPP-Grid broker can continue to use the latest available version.

Negative consequences:

- Additional effort is required for creating and testing new components.
- Angulars component features such as style scoping are not fully utilised.

8. Frontend Structure

We must establish a clean architectural structure to keep the frontend maintainable.

8.1. Considered Options

- Monolyth: All components and services in a single module to support fast-prototyping
- Module-Based: Hierarchical Module-Structure, grouping related components.

8.2. Decision Outcome

We decided on the module-based option as it follows software engineering best practices for long-lived projects. We accept a slower starting phase to benefit in the long-term from a clean architecture.

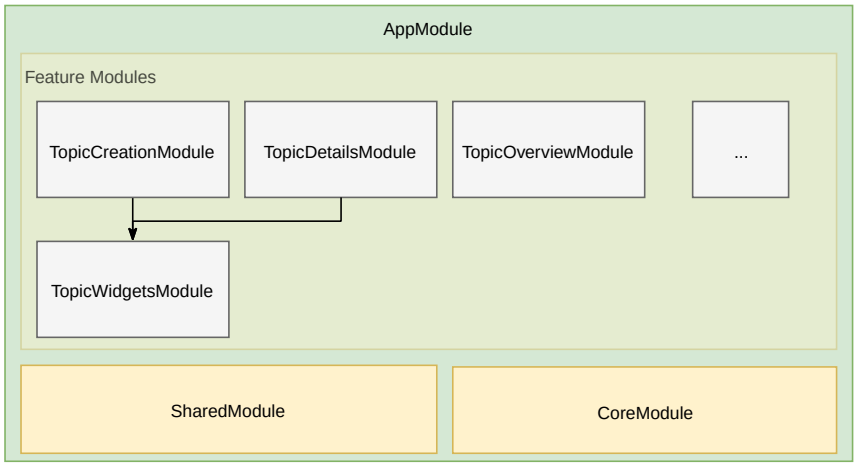
Following the Angular Style Guide, we decided on two primary modules: `core`, `shared` as well as additional widget and feature modules.

Each module can have submodules to refine the structure even further.

Because the current scope is rather small, all modules are loaded eagerly.

The `core` module contains the XMPP connection setup, authentication and other functionality that is required at all times.

The `shared` module contains simple components that do not depend on any services but are just plain components used to render common structures, e.g. buttons.



9. XMPP Client Library

To communicate via XMPP from the client via Websockets or Bosh an existing client library shall be used to benefit from an existing implementation.

Primarily, the following aspects are relevant for the decision, ordered by priority:

1. Community Support: How well is the project maintained
2. Plugin API: Is it easy to create or replace existing plugins
3. TypeScript binding: Are there existing TypeScript bindings?

9.1. Considered Options

- Stanza.io
- strophe.js

9.2. Decision Outcome

Chosen option: stanza.io, because it offers solid community support and provides a clean plugin API with minimal dependencies. Although strophe.js has a more active community, some plugins have dependencies on jQuery and are implemented in CoffeeScript. That is the main reason why we selected Stanza.io over strophe.js.

To fully benefit from TypeScript, we could implement the TypeScript bindings. https://www.typescriptlang.org/docs/handbook/declaration-files/introduction.html

9.3. Evaluations of the Options

9.3.1. Stanza.io

1. Community Support
    - > 20 Contributors
    - primarily maintained by one person
    - Project receives frequent updates since 2014
    - Used by 40 Repos and 3 Packages (2018-04-10)
    - > 10 Pull-Requests closed/merged since 2017-01-01
2. Plugin API
    - New plugins can easily be added
    - Existing plugins can be replaced
3. TypeScript binding
    - No TypeScript bindings exist

9.3.2. strophe.js

1. Community Support (only core repository)
    - > 50 Contributors

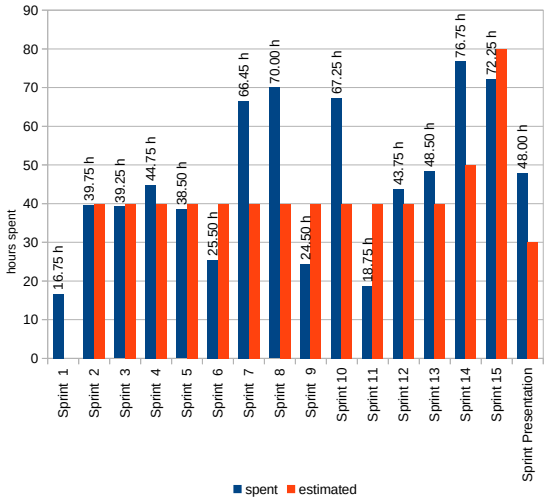
- minor fixes in the last year by different contributors
- Project regularly updated since 2008
- Used by >90 Repos and >20 Packages (2018-04-10)
- > 25 Pull-Requests closed/merged since 2017-01-01

2. Plugin API
   - Some Plugins depend on jQuery
   - Minimal core allows to add and replace existing plugins simply
   - Many plugins are written in CoffeeScript (whereas the core is in JavaScript)
3. TypeScript binding
   - Bindings for the core exist (not tested!), see @types/strophe on NPM

A.4 Time Accounting

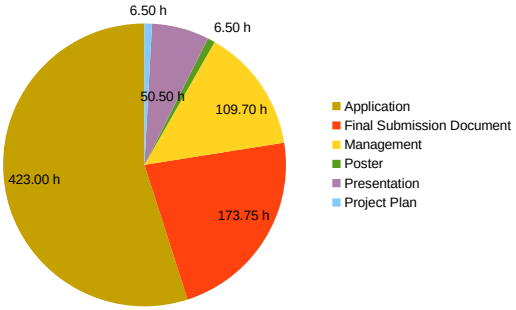
Time Accounting

Time Spent per Sprint



We spent on average 46.29 hours per sprint conforms to the usually intended 40 hours. The work, however, is not distributed equally over the sprints.

Time Spent per Component

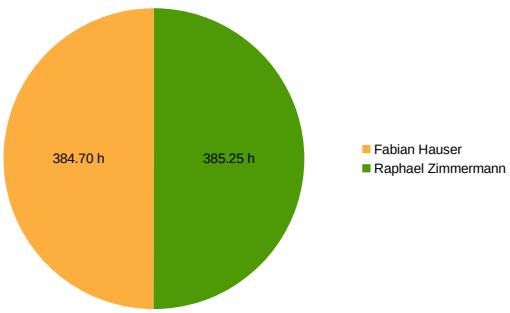


As intended, we spent the most time on the application. The management that includes sprint planning, standup and advisory meetings, as well as infrastructure tasks, took approximately a seventh of the projects time budget. We also spent a lot of time on the final submission document, as it is an essential part of the bachelor thesis.

Estimated Cost

Using the hourly wage from the module Software Engineering 2, the total project costs would be 66'408.00 CHF in practice.

Time Spent per Person



A.5 Meeting Minutes

1. Meeting 2018-02-19

1.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

1.2. Agenda

1. Administrative tasks
2. Where to start
3. Date and time for the next meeting
    - 2018-02-26, 09:00 in SFFs office

1.3. Discussions / Decisions

1. Administrative tasks
    - All documentation artifacts will be published on the project website.
    - We will use JIRA for project planning.
    - We will decide later, which continous integration tools we use.
        - The decision must allow the INS to take over the project after the BA.
        - reasons for the decision must be documented.
    - Decisions on the programming language and frameworks are made later.
        - Our experience and productivity in a given eco system must be considered as well.
2. Where to start
    - Read the draft XMPP for Security Information Exchange.
    - Learn more about XMPP (Read the specs and the mentioned XEPs).
    - IODEF payloads are not the main focus of this project.
    - It would be nice if a first draft is ready for the IETF Hackathon, starting on March 17.
    - Main goals of the project:
        - Design a solid architecture (Openfire plugin or standalone?).
        - Implement the requirements according to the IETF Draft:
            - Vanilla XMPP with Discovery und Publish/Subscribe XEPs.
            - Define Topics/Nodes and manage permissions.
            - Administrative utilities such as purge, list topics, show number of items, identifier etc.
        - Most is already implemented in the form of a proof of concept.
3. Date and time for the next meeting:

- 2018-02-26, 09:00 in SFFs office

1.4. Upcoming absences

no upcoming absences

2. Meeting 2018-02-26

2.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

2.2. Agenda

1. Open questions regarding the task
2. Date for oral exam
3. Is there a presentation during the semester?
4. Date and time for the next meeting

2.3. Discussions / Decisions

1. Open questions regarding the task
    - Some requirements can influence the chosen architecture fundamentally (e.g. to write a "bot", a component or a plugin)
        - a "normal" user (in the "bot" or the component variant) might have limited access to some node. According to SFF, this issue can be ignored because, in a real-world application, such behaviour should be limited by strict policies.
        - SFF emphasises that the authentication mechanisms use must be robust and certificate based.
    - Next steps:
        - Write User stories
        - Draw basic architecture in C4-Diagrams
        - Risk analysis (e.g. abuse cases)
        - Evaluation of XMPP servers and libraries. SFF notes that we should not spend too much time evaluating the server and assume that OpenFire fulfils most requirements.
    - Issues to address in the XMPP servers and library evaluation:
        - Can an administrator restrict users to create new topics
        - Recommended features can be checked queried using service discovery. We can also check for undesired configurations (e.g. everyone can publish)
        - Ensure that the libraries support all required functionality, especially authentication!
        - Assess the performance and usability of the libraries
    - It is desirable if the service runs is "always on", e.g. to answer subscription requests. However, this is not strictly required according to SFF.
    - Any kind of Interface is conceivable, a web interface, however, is very flexible. The core functionality does not have to be available separately.
    - The scope of the website is fine according to SFF.

2. License
    - GNU-FDL for the documentation is fine
    - The license for the code will be AGPL but might change depending on the frameworks we use
3. Is there a presentation during the semester?
    - An interim with the internal co-examiner will be carried out.
    - The primary purpose of this presentation is to get familiar with the requirements of the co-examiner (testing, documentation, protocols etc.)
    - Should be carried out if a small demo is ready
4. Date for oral exam
    - If possible, the oral exam will be carried out in early July.
    - We will continuously complete parts of the document to reduce the examination efforts
5. Date and time for the next meeting

2.4. Upcoming absences

no upcoming absences

3. Meeting 2018-03-05

3.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

3.2. Agenda

1. Discuss and clarify Requirements/Wireframes
2. Current state of the Task Description
3. Date and time for the next meeting

3.3. Discussions / Decisions

1. Discuss and clarify Requirements/Wireframes
    - We discussed the open issues in the current requirements draft.
        - The application is meant to run in production and must, therefore, meet strict security requirements
            - Especially TLS >= 1.2 and certificate-based
            - The existing python scripts were the prototype and this project is intended to be a proper implementation
        - Response time is not very crucial for this application as it is designed for maintenance work and not real-time interventions.
        - We must propose an authentication model, e.g. using TLS mutual authentication and exactly one broker user.
        - SFF notes that SASL is quite complex. We might not need it although the IETF draft explicitly requires it.
        - The number of consumers and producers depends strongly on the concrete application. In most cases, however, there will be much more publishers than subscribers. SFF gave the following estimates as reference values.
            - > 1000 Producers
            - 100 Subscribers
            - 1-4 Toplevel Topics
            - > 1000 Subtopics (eg. one for every publisher)
            - > 10000 persisted items (note that these items might contain large payloads)
        - We plan to include search, filtering and paging functionality for most listings. We will include these features in the next requirements draft.
        - SFF points out that subtopics are missing in the current draft as well as the wireframes. We will include this in the next requirements draft.

        - SFF notes that a role concept might be needed to simplify the administration. We might be able to use existing XMPP features for this (e.g. XEP-0144).
        - According to SFF, Subscription Requests and Validation are nice to have but have low priority.
            - Instead of validation, unsupported functionality should not be visible
        - For deleting persisted items, SFF suggests a "bulk delete" functionality, which allows administrators to delete all items that match certain criteria.
        - SFF prefers a standalone application over a server plugin to reduce coupling.
        - The implementation language is not of paramount importance to SFF although he prefers Python or Java. A single page app written in JavaScript using a Python backend is also a viable option for him.
            - We should not rely on too many third-party libraries that save us several hours during the project but might require extra maintenance effort in the future (SFF gave a Django paging extension as an example)
2. Current State of the Task Description
    - SFF will complete the task description after he receives the revised user stories and wireframes.
3. Date and time for the next meeting
    - 2018-03-12, 09:00 in SFFs office

3.4. Upcoming absences

- No weekly meeting on 2018-03-19 (SFF is absent)

4. Meeting 2018-03-12

4.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

4.2. Agenda

1. Discuss Requirements / Wireframes
2. Architectural Decisions
    - Architectural Decision: Roles
    - Architectural Decision: XMPP over Websockets/BOSH/Backend
    - Architectural Decision: Client Framework
3. Current State of the Task Description
4. Date and time for the next meeting

4.3. Discussions / Decisions

1. Discuss Requirements / Wireframes
    - We will send SFF the updated wireframes and requirements.
    - Next week, SFF will accept/reject the current state as the target set of requirements for the thesis.
    - SFF considers the current wireframes as good.
2. Architectural Decisions
    - Architectural Decisions Template
        - SFF appreciates the current format.
        - We must give more context for non-experts (e.g. what is an XMPP-Bot?).
    - Architectural Decision: Roles
        - We discussed the pros and cons as described in the Architectural Decision.
        - The biggest downside is the limited Openfire support as well as the deferred state of XEP-0248.
        - SFF ratifies this solution.
    - Architectural Decision: XMPP over Websockets/BOSH/Backend
        - SFF says, that this is an interesting approach to experiment with.
        - We must conduct experiments to ensure that all functionality indeed works as expected.
        - SFF says, that from a security aspect, a new interface is not that bad as long as it uses secure authentication mechanisms such as TLS.
        - If we decide to use WebSockets/BOSH, we should investigate how an adapter could be implemented.

    - Architectural Decision: Client Framework (e.g. React, Vue, Angular)
        - For SFF, the most important criteria is to not rely on too many third-party dependencies.
        - Otherwise, we should use the best tool for the task.
        - The institute has not preferred framework.
3. Current State of the Task Description
    - Will be finished until the next meeting, based on our revised requirements.
    - SFF will send the current draft.
4. Date and time for the next meeting
    - No weekly meeting on 2018-03-19
    - Next weekly meeting is on 2018-03-23 10:00

4.4. Upcoming absences

- No weekly meeting on 2018-03-19 (SFF is absent)

5. Meeting 2018-03-23

5.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

5.2. Agenda

1. Discuss Requirements / Wireframes
2. Current State of the Task Description
3. The current state of co-examinator
4. "Broker" Term
5. Architectural analysis with an industry expert (XGB-97)
6. Date and time for the next meeting

5.3. Discussions / Decisions

1. Discuss Requirements / Wireframes
   - SFF accepts the current state of the wireframes.
2. Current State of the Task Description
   - We collectively declared the Task Description final.
   - SFF suggests to move it up in the document before the abstract (XGB-105).
3. The current state of co-examinator
   - Prof. Dr Thomas Bocek is the co-examinator of this thesis.
   - We will organise the interim presentation in 2-5 weeks from now, ideally combined with a weekly meeting and not on Tuesdays.
4. "Broker" Term
   - While reading the XMPP-Grid IETF draft, we were confused by the definition of "Broker", and whether it applies to our application.
   - SFF sees no discrepancy. We might not implement the complete broker but at least parts of it.
5. Architectural security analysis with an industry expert (XGB-97)
   - SFF suggests conducting the security analysis with Tobias Brunner on a Monday.
6. Date and time for the next meeting
   - No weekly meeting on 2018-03-26 (SFF absent)
   - No weekly meeting on 2018-04-02 (Easter Monday)
   - Next weekly meeting on 2018-04-09

5.4. Upcoming absences

- No weekly meeting on 2018-03-26 (SFF absent)
- No weekly meeting on 2018-04-02 (Easter Monday)

6. Meeting 2018-04-09

6.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

6.2. Agenda

1. Current State and Next Steps
2. Administrative Tasks
3. Date and time for the next meeting

6.3. Discussions / Decisions

1. Current State and Next Steps
    - rzimmerm presents a demo of the current client
    - fhauser explains the TLS client certificate issues we encountered in the last weeks.
    - SFF suggest to make and document assumptions (eg. nginx and openfire are co-located) and begin with the implementation
    - SFF suspects issues with the XMPP client libraries
2. Administrative Tasks
    - Intermediate Presentation: SFF suggest we do this in the next three weeks with a working demo including XMPP
    - Final Presentation:
    - 2018-06-27 is the ideal date for rzimmerm and fhauser
    - We will contact Prof. Dr. Thomas Bocek concerning this date
    - SFF will contact Dr. Ralf Hauser
3. Date and time for the next meeting
    - Next weekly meeting on 2018-04-16

6.4. Upcoming absences

none

7. Meeting 2018-04-16

7.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF
- Tobias Brunner

Minute Taker: rzimmerm

7.2. Agenda

1. Administrative Tasks
2. Demo
3. Architectural Security Review
4. Date and time for the next meeting

7.3. Discussions / Decisions

1. Administrative Tasks
   - intermediate presentation
     - No response yet from Prof. Dr. Thomas Bocek. We will ask him today in the exercises session.
     - SFF is unavailable next Monday and suggests to do the presentation without him
   - Final presentation
     - No response yet from Prof. Dr. Thomas Bocek. We will ask him today in the exercises session.
     - SFF will contact Dr. Ralf Hauser until next week
2. Demo
   - fhauser demonstrates a working XMPP connection from the Angular application using TLS with Client certificates
   - rzimmerm demonstrates the generic data forms implementation
   - We will enable SFF to try it out with his setup next Week.
3. Architectural Security Review
   - Tobias Brunner notes, that TLS Client Certificates without an Origin policy can be problematic
     - When using WebSockets or Bosh, the XMPP server (or Nginx) must verify the Origin header
4. Date and time for the next meeting
   - SFF is unavailable next Monday.
   - The meeting is postponed to 2018-04-26

7.4. Upcoming absences

- SFF: 2018-04-23

- fhauser: 2018-04-30

8. Meeting 2018-04-26

8.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

8.2. Agenda

1. Administrative Tasks
2. How to Test-Drive
3. Presentation Date
4. Review: Issues we encountered
5. Date and time for the next meeting

8.3. Discussions / Decisions

1. Administrative Tasks
   - The Interim presentation took place on Monday
2. How to Test-Drive
   - SFF wants to try out the application as discussed last week
   - To use an existing XMPP server, we recommend using the development setup with another upstream XMPP server
     - The CA and Client Certificate for connecting to the XMPP server must be replaced
     - The upstream URL must be replaced
   - We will publish an alpha release this week
3. Presentation Date
   - Prof. Dr. Thomas Bocek is available on 2018-06-27
   - SFF will clarify if this date works for the other groups and the expert as well.
4. Review: Issues we encountered
   - We were missing some functionality in the XMPP standards
     - loading all available configuration options before the creation of a topic
     - Lost updates when updating the topic configuration (eg. `max-items` and `node_type`)
   - We decided to only focus on the features as Openfire support it
5. Date and time for the next meeting
   - fhauser is absent on 2018-04-30 (Zivilschutz)
   - The meeting is postponed to 2018-05-07

8.4. Upcoming absences

- fhauser: 2018-04-30 (Zivilschutz)

9. Meeting 2018-05-07

9.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

9.2. Agenda

1. Review: Test-Drive
2. Prioritisation: On what should we focus?
3. Presentation Date
4. Date and time for the next meeting

9.3. Discussions / Decisions

1. Review: Test-Drive
    - SFF has not been able to try it out yet.
    - SFF wants to have a look at it tomorrow
2. Prioritisation: On what should we focus?
    - We discussed all open user stories
    - The core functionality is (almost) complete
        - Subtopics/Parent Collections view is missing
        - Subscription view is missing
        - Persisted Items
    - SFF wants us to add persisted items next
        - Autocomplete has a low priority for SFF
    - We also discussed the "problematic" user stories, e.g. that a logout mechanism is not possible.
        - We agree that they are no show-stoppers.
3. Presentation Date
    - Prof. Dr. Thomas Bocek is available on 2018-06-27
    - SFF will clarify if this date works for the other groups and the expert as well.
4. Date and time for the next meeting
    - SFF is absent on 2018-04-14
    - The meeting is postponed to 2018-05-17, 09:00

9.4. Upcoming absences

- SFF: 2018-04-14
- Whitmonday: 2018-04-21

10. Meeting 2018-05-17

10.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

10.2. Agenda

1. Current State & Issues
2. Review: Test-Drive
3. Presentation Date
4. Date and time for the next meeting

10.3. Discussions / Decisions

1. Current State & Issues
   - We showed SFF the demo of the persisted items view
   - We discussed the limitations of the XEPs (XEP-0060, XEP-0059) and their implication on the feature set that we can implement
     - For SFF it's not the highest priority to implement these features (e.g. autocomplete, search and filtering)
2. Review: Test-Drive
   - SFF was quite busy and has not had the time yet to try it out
3. Presentation Date
   - SFF will contact Dr. Ralf Hauser today
4. Date and time for the next meeting
   - Next weekly meeting on 2018-04-28
   - If SFF has feedback concerning the test drive, a short-dated meeting will be carried out on 2018-04-24.

10.4. Upcoming absences

- Whitmonday: 2018-05-21

11. Meeting 2018-05-28

11.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

11.2. Agenda

1. Review: Test-Drive
2. Presentation
3. Date and time for the next meeting

11.3. Discussions / Decisions

1. Review: Test-Drive
    - The location of the install instruction was not clear
    - Having a documentation repository can be confusing
    - We will add a note admonition to the documentation page
2. Presentation
    - The date and time for the presentation is fixed: 2018-06-27T15:00
    - We will start with a presentation (20-30 minutes) covering…
        - the problem
        - our solution
        - the biggest challenges
        - special/notable efforts
        - demo
3. Date and time for the next meeting
    - Next weekly meeting on 2018-06-04 09:00

11.4. Upcoming absences

None

12. Meeting 2018-06-04

12.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

12.2. Agenda

1. Review Test-Drive
2. Discuss Abstract
3. Project state
4. Date and time for the next meeting

12.3. Discussions / Decisions

1. Review Test-Drive
    - SFF noted the following issues:
        - The application is slow in SFFs setup, which might be improved by modifying the TLS options (`proxy_ssl_session_reuse` and `proxy_request_buffering`). Issue: XGB-217.
        - Timeouts occur even if manually configuring a bigger value. Issue: XGB-216.
        - The page size is too small and should be configurable or at least 50 elements. XGB-217.
        - The error message when returning 403 can be improved ("connection lost").
            - As discussed, this can be fixed properly by configuring nginx differently.
2. Project state
    - We added a new installation guide.
    - We wrote more documentation.
        - We will send a draft on Thursday to SFF for review.
    - We will address the issues from the test drive.
    - We will extend the e2e tests if we have enough time.
3. Discuss Abstract
    - SFF suggests extending the implementation section.
        - Discuss Architecture in one paragraph and the functionality in another one.
        - clarify that it's a 100% web-based application that (usually) requires a proxy.
        - Try to "sell" the project.
        - Can be longer.
        - The target audience has generic software know-how (e.g. knows what JS is) but has no expertise in XMPP.

- SFF says, that the management summary is not necessary.

4. Date and time for the next meeting
    - Will be arranged at short notice if needed.

12.4. Upcoming absences

None

13. Meeting 2018-06-13

13.1. Attendees

- Fabian Hauser, fhauser
- Raphael Zimmermann, rzimmerm
- Prof. Dr. Andreas Steffen, SFF

Minute Taker: rzimmerm

13.2. Agenda

1. Signing of the usage rights
2. Abstract status
3. Review of the final submission document draft
4. Electronic submission
5. Project transfer

13.3. Discussions / Decisions

1. Signing of the usage rights
   - SFF signed the usage rights.
2. Abstract status
   - SFF accepted and forwarded our abstract to the administration.
3. Review of the final submission document draft
   - SFF notes that he would have prefered parts of the appendix (eg. architectural decisions) in the main content.
   - The IODEF parts are not 100% accurate anymore and can safely be removed.
4. Electronic submission
   - According to SFF, it is sufficient to upload the artifacts via the official tools.
5. Project transfer
   - We will install SFF as organisation administrator on our GitHub repository.
   - Further steps can be arranged bilaterally.

A.6 Requirements

The following sections describe the primary requirements in the form of user stories [4]. Figure *1* shows an overview of the primary use stories.

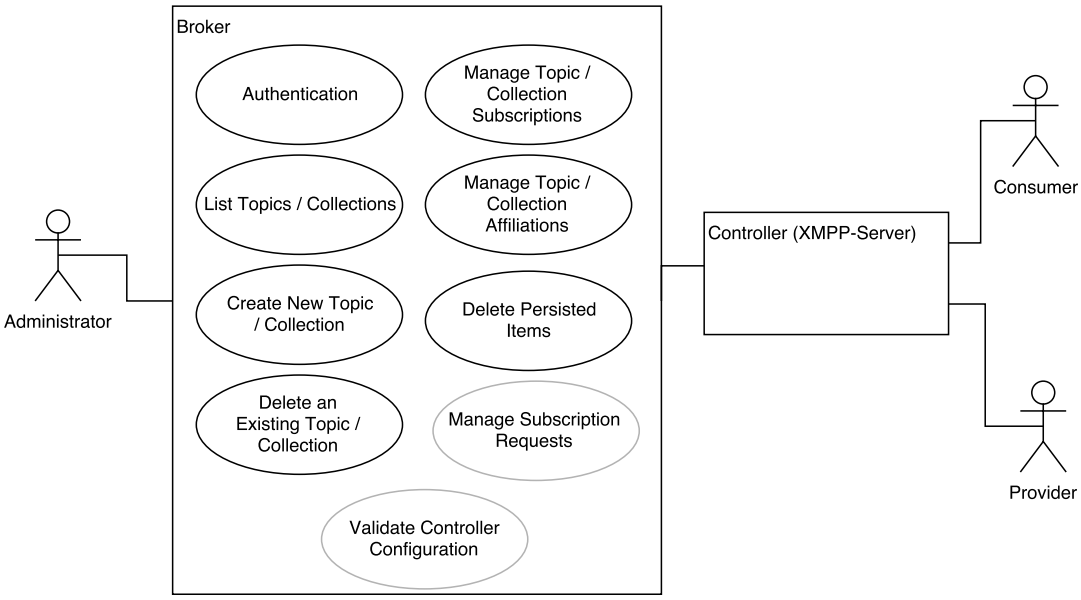


FIGURE 1: UML use case diagram presenting an overview of the primary user stories.

A.6.1 Authentication

A.6.1.1 Login

As an Administrator,
I want to log in
- preferably using an existing client *TLS* certificate -
so that only I can inspect and manage topics.

A.6.1.2 Secure *XMPP* Authentication

As an Administrator concerned with security requirements,
I want to use either *SASL EXTERNAL* or *SASL SCRAM* mechanism for authentication -

- preferably the SCRAM-SHA-256-PLUS variant and
- preferably using mutual certificate-based authentication including revocation status checking

- so that the controller is fully compatible with the *XMPP-Grid standard* [8].
To achieve this goal, I am willing to accept:

- More costly and less user friendly authentication
- limited compatibility of supported *XMPP* servers

A.6.1.3 Secure *XMPP* Connection

As an Administrator concerned with security requirements,
I want to use minimally *TLS* 1.2 [RFC5246] to communicate with the *XMPP* server at all times
to achieve maximal security and compatibility with the *XMPP-Grid standard* [8].

A.6.1.4 Secure Connection

As an Administrator concerned with security requirements,
I want to use minimally *TLS* 1.2 [RFC5246] to communicate with the *broker*
to achieve maximal security.

A.6.1.5 Multiple Administrators

As an Administrator,
I want to grant access to administrators
so that they can also manage the application.

A.6.1.6 Audit Trail

As an Administrator concerned with security requirements,
I want to be able to access an audit log
- preferably using existing *XMPP* mechanisms -
so that I can reconstruct what other Administrations did on the controller.

A.6.1.7 Logout

As an Administrator,
I want to log out
so that I can terminate a session.

A.6.2 List Topics and Collections

A.6.2.1 List All Topics

As an Administrator,
I want to see a list of all topics of the associated controller
so that I can quickly assimilate which topics exist.

A.6.2.2 List All Top-Level-Collections

As an Administrator,
I want to see a list of all top-level-collections of the associated controller
so that I can quickly assimilate which collections exist.

A.6.2.3 List All Parent-Collections of a Topic

As an Administrator,
I want to see a list of all transitive parent collections that contain a given topic
so that I can quickly assimilate in which collections items are published.

A.6.2.4 List All Subtopics and Subcollection of a Collection

As an Administrator,
I want to see a list of all collections and topics that a given collection contains
so that I can quickly assimilate the collection hierarchy.

A.6.2.5 List Available topics With Limited Access (optional)

As an Administrator,
I want to see a list of all topics of the associated controller to which I have limited access to,
to simplify troubleshooting and locate errors.

A.6.2.6 List Available collections With Limited Access (optional)

As an Administrator,
I want to see a list of all collections of the associated controller to which I have limited access to,
to simplify troubleshooting and locate errors.

A.6.2.7 Topic and Collection Paging

As an Administrator,
I want to be able to page through any set of collection/topic with more than 10 Items
so that I can work with more than 1000 collections and topics more effectively.

A.6.2.8 Topic and Collection Name Filter

As an Administrator,
I want to be able to quickly filter any set of collections/topics with more than 10 Items
so that I can work with more than 1000 collections and topics more effectively.

A.6.3 Create a New Topic

As an Administrator,
I want to create a new topic on the associated controller
so that I am not tied to a fixed set of topics.

A.6.4 Create a New Collection

As an Administrator,
I want to create a new collection on the associated controller
so that I can flexibly patch topics together.

A.6.4.1 Override Default Topic Configuration

As an Administrator in the process of creating a new topic,
I want to override the default configuration (e.g. the affiliations)
so that I can restrict access and provide reasonable defaults.

A.6.4.2 Override Default Collection Configuration

As an Administrator in the process of creating a new collection,
I want to override the default configuration (e.g. the affiliations)
so that I can restrict access and provide reasonable defaults.

A.6.4.3 Initial topic Consumers and Providers

As an Administrator in the process of creating a new topic,
I want to specify an initial set of consumers and providers
so that I can restrict access to that topic and provide reasonable defaults.

A.6.4.4 Initial Collection Consumers

As an Administrator in the process of creating a new collection,
I want to specify an initial set of consumers
so that I can restrict access to that collection and provide reasonable defaults.

A.6.5 Delete an Existing Topic

As an Administrator,
I want to delete an existing topic on the associated controller
so that I can get rid of obsolete topics.

A.6.6 Delete an Existing Collection

As an Administrator,
I want to delete an existing collection on the associated controller
so that I can get rid of obsolete collections.

A.6.6.1 Fault Prevention On Topic-Delete

As an Administrator in the process of deleting a topic,
I want a mechanism to prevent me from deleting the wrong topic on the associated controller
(e.g. require me to enter the name of the topic manually).

A.6.6.2 Fault Prevention On Collection-Delete

As an Administrator in the process of deleting a collection,
I want a mechanism to prevent me from deleting the wrong collection on the associated controller
(e.g. require me to enter the name of the collection manually).

A.6.7 Manage Topic/Collection Subscriptions

A.6.7.1 List Consumers

As an Administrator,
I want to list all consumers (including their JIDs) of a given topic/collection on the associated controller,
so that I can verify that specific consumers are subscribed, and others are not.

A.6.7.2 Inspect Detailed Subscription Configuration

As an Administrator,
I want to inspect the detailed topic/collection subscription configuration of a given consumer,
so that I can reproduce and reason about the receipt of data on that consumer and find potential misconfiguration.

A.6.7.3 Partially Modify Subscription Configuration

As an Administrator,
I want to modify parts of the topic/collection subscription configuration of a given consumer,
so that I can fix misconfiguration.

A.6.7.4 Unsubscribe Consumer

As an Administrator,
I want to manually unsubscribe a specific consumer from a particular topic/collection on the associated controller,
so that I can remove obsolete or undesired subscriptions.

A.6.7.5 Subscribe Consumer

As an Administrator,
I want to manually subscribe a specific consumer on a particular topic/collection on the associated controller,
so that I can faster setup and manage consumers.

A.6.8 Manage Topic Affiliations

A.6.8.1 Inspect Affiliations

As an Administrator,
I want to list all Affiliations (JID and "Role") for a particular topic/collection on the associated controller
so that I can find potential misconfiguration.

A.6.8.2 Modify Affiliations

As an Administrator,
I want to modify the Affiliation ("Role") of a given JID for a particular topic/collection on the associated controller
so that I can fix potential misconfiguration.

A.6.8.3 Fault Prevention When Modifying My Affiliation

As an Administrator in the process of modifying my Affiliation for a particular topic/collection on the associated controller,
I want a mechanism to prevent me from accidentally downgrading my rights.

A.6.8.4 Meaningful Error For Topics/Collection With Limited Access

As an Administrator,
I want to receive a meaningful error message when inspecting a topic/collection to which I have limited access
so that I can quickly comprehend why the configuration options are limited.

A.6.9 Manage Persisted Items of a Topic

A.6.9.1 Inspect Persisted Items

As an Administrator,
I want to list all persisted items for a particular topic on the associated controller
so that I can get an overview and check for misconfiguration.

A.6.9.2 Filter Persisted Items

As an Administrator,
I want to be able to filter all persisted items of a specific topic by

- the timestamp of its publication
- the publishers JID

so that I can work with more than 10000 persisted items more effectively.

A.6.9.3 Paged Persisted Items

As an Administrator working with filtered persisted items,
I want to be able to page through the resulting items
- given that this feature is supported by the associated controller -
so that I can work with more than 10000 persisted items more effectively.

A.6.9.4 Delete a Persisted Items From a Topic

As an Administrator,
I want to delete a particular persisted item from a specific topic
- given that this feature is supported by the associated controller -
so that I can clean up test items and remove obsolete or corrupted items.

A.6.9.5 Purge All Persisted Items From a Topic

As an Administrator,
I want to purge persisted items from a specific topic
- given that this feature is supported by the associated controller -
so that I can clean up test items and remove obsolete or corrupted items.

A.6.9.6 Delete Set of Persisted Items From a Topic (optional)

As an Administrator,
I want to delete a set of persisted item that match a given criteria from a specific topic
- given that this feature is supported by the associated controller -
so that I can clean up test items and remove obsolete or corrupted items.

A.6.10 Manage Subscription Requests (optional)

A.6.10.1 List Subscription Request

As an Administrator,
I want to list pending subscription requests for a given topic
- given that this feature is supported by the associated controller -
so that I can quickly assimilate pending requests.

A.6.10.2 Accept Subscription Request

As an Administrator,
I want to accept a pending subscription request for a given topic
- given that this feature is supported by the associated controller -
to enable more dynamic access models than just maintaining a black- or whitelist.

A.6.10.3 Reject Subscription Request

As an Administrator,
I want to reject a pending subscription request for a given topic
- given that this feature is supported by the associated controller -
so that I can deny user access in accordance with the *XMPP* standards.

A.6.11 Validate Controller Configuration (optional)

A.6.11.1 Validate Supported XEPs Configurations

As an Administrator,
I want to validate that a minimum set of XEPs are supported by the associated controller
so that I can quickly identify incompatibilities.

A.6.11.2 Validate Optional XEP Implementations

As an Administrator,
I want to validate that the required features that are marked as optional or recommended in the XEPs are implemented by the associated controller
so that I can quickly identify incompatibilities.

A.7 Wireframes

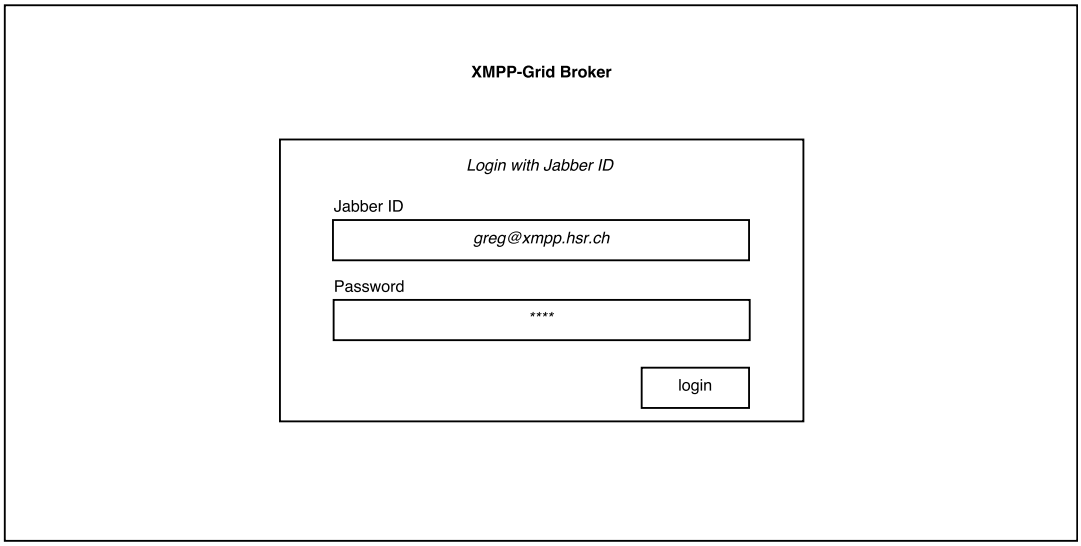


FIGURE 2: Login-screen wireframe

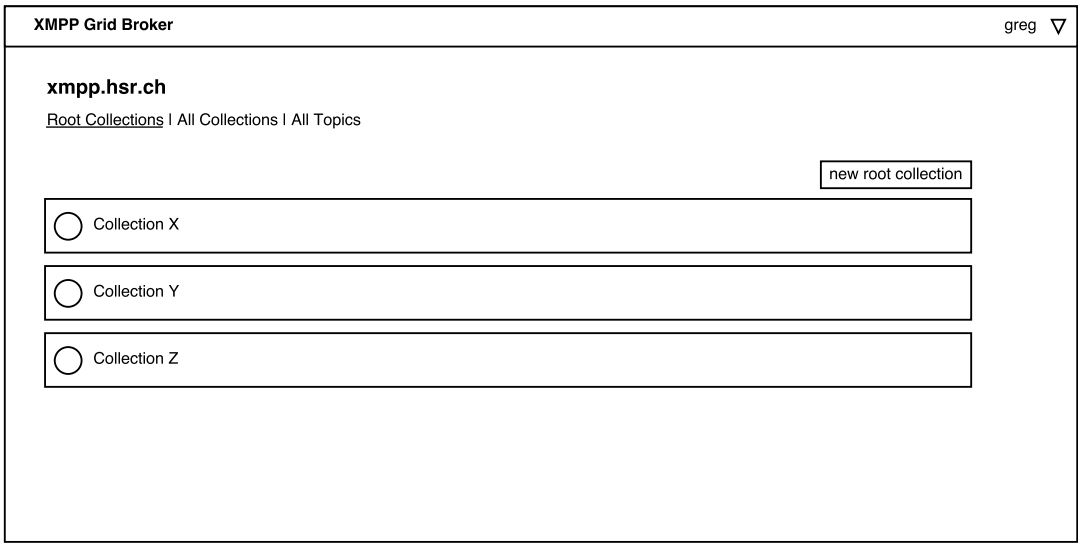


FIGURE 3: Controller overview wireframe

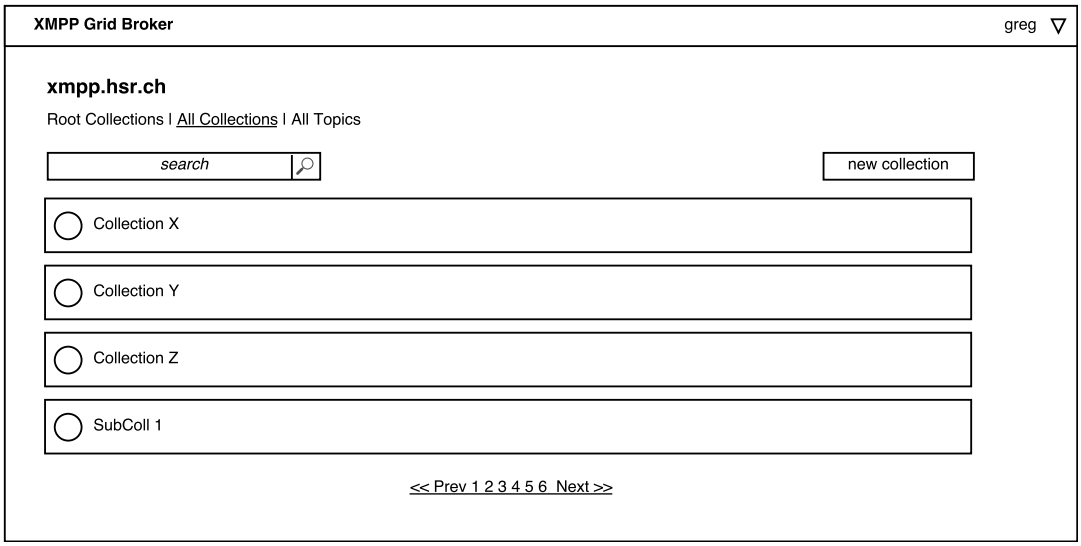


FIGURE 4: All collections wireframe

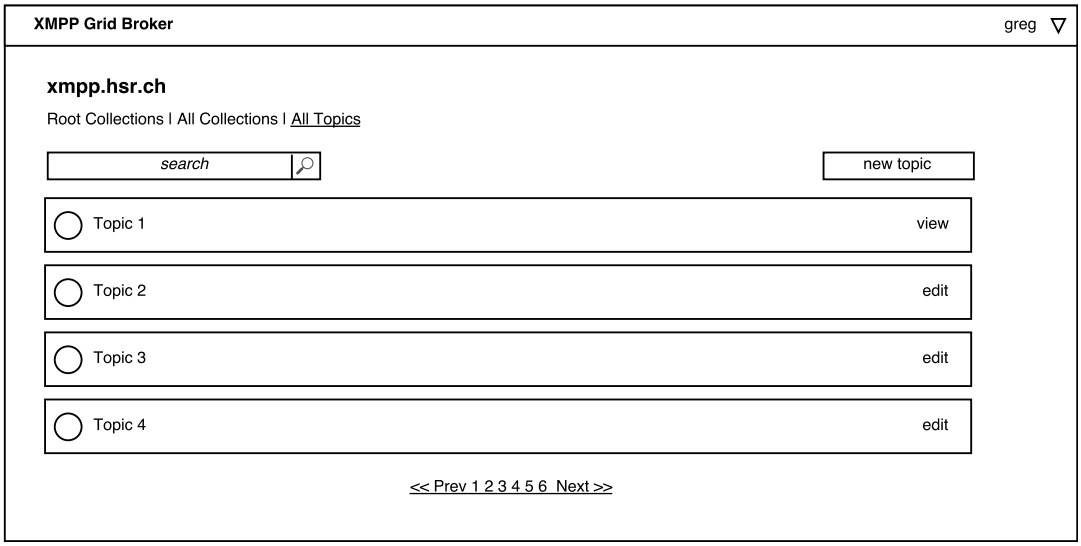


FIGURE 5: All topics wireframe

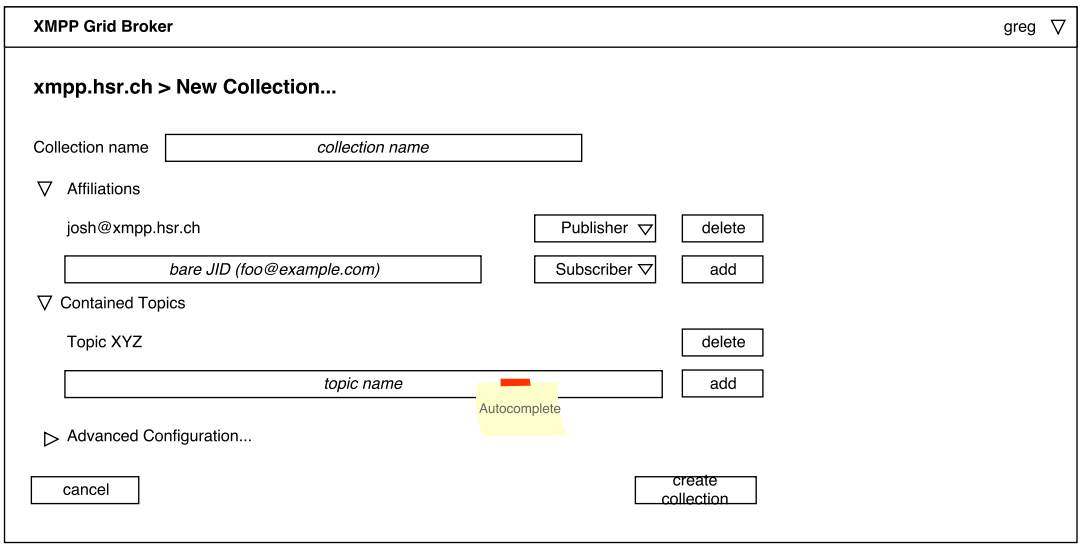


FIGURE 6: New collection wireframe

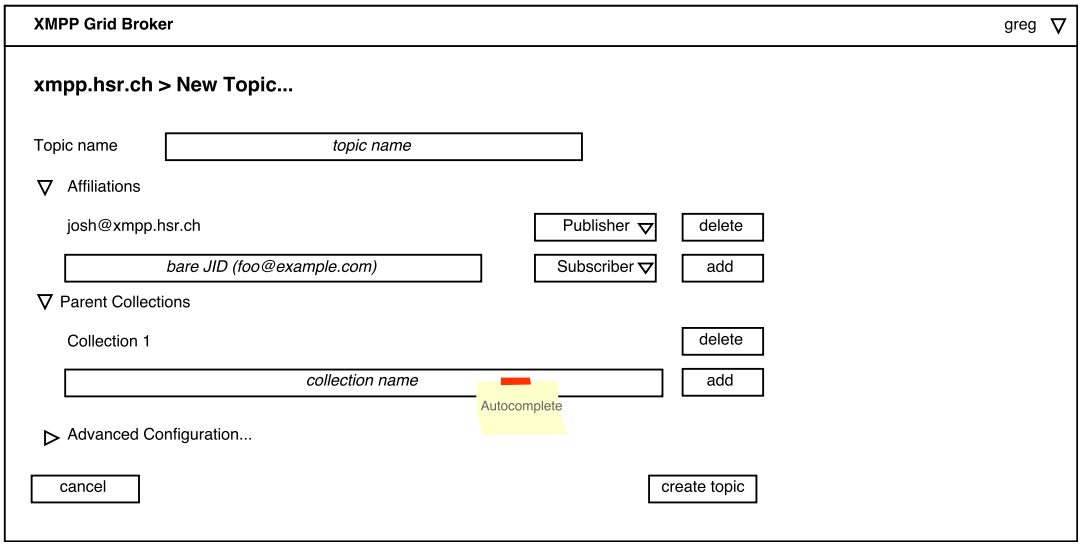


FIGURE 7: New topic wireframe

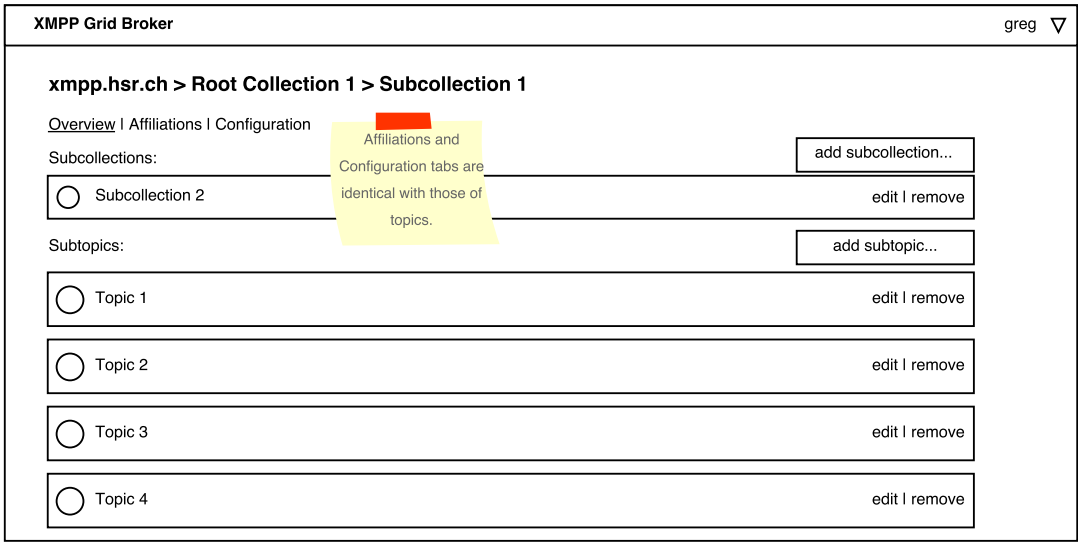


FIGURE 8: Collection overview wireframe

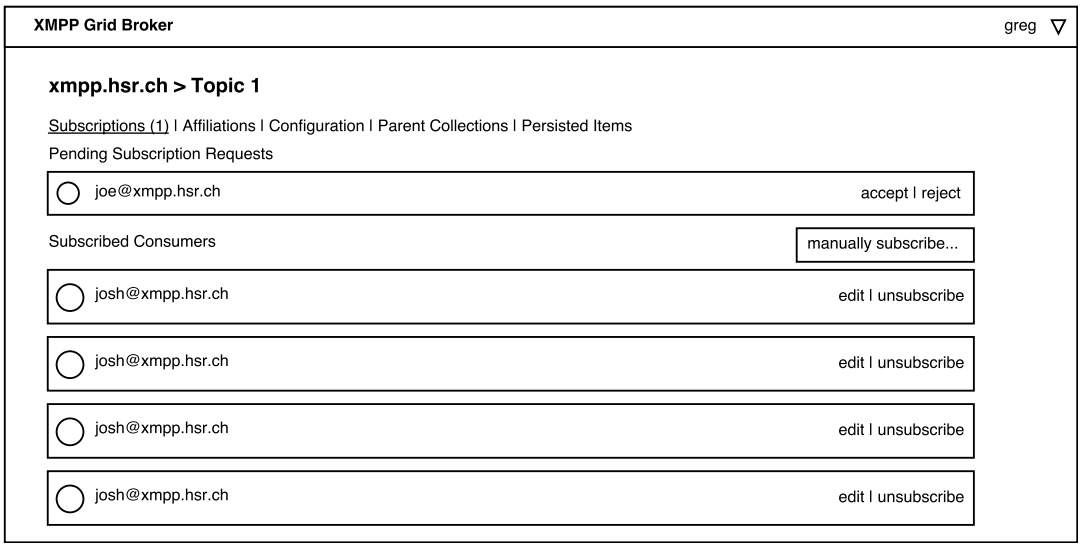


FIGURE 9: Topic overview wireframe

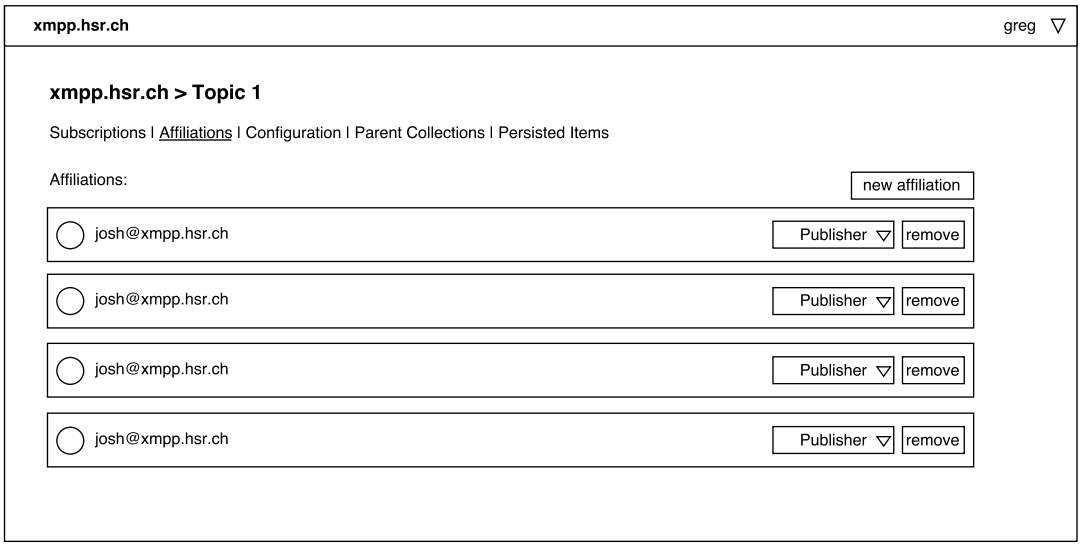


FIGURE 10: Topic/Collection affiliations wireframe

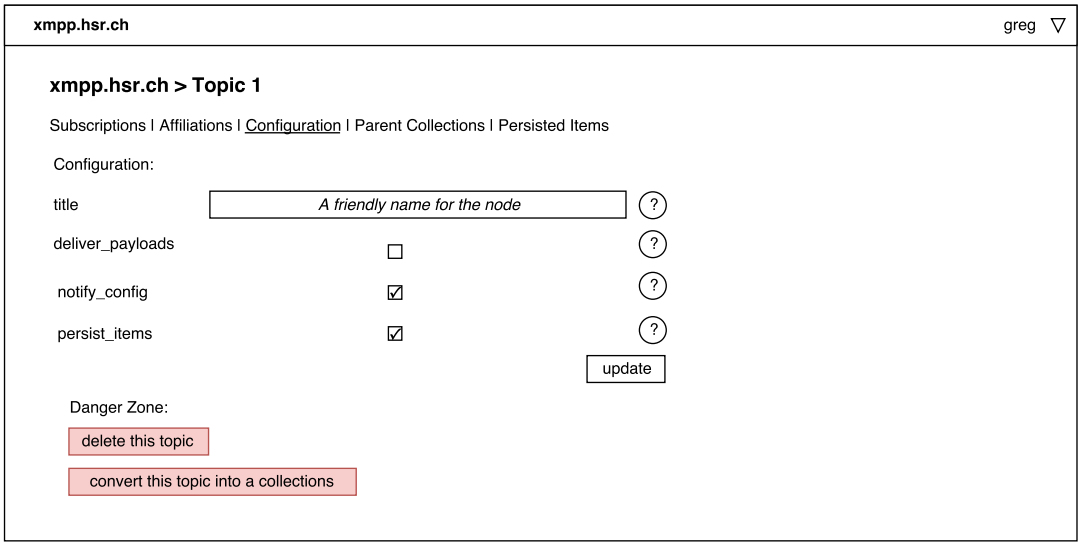


FIGURE 11: Topic/Collection configuration wireframe

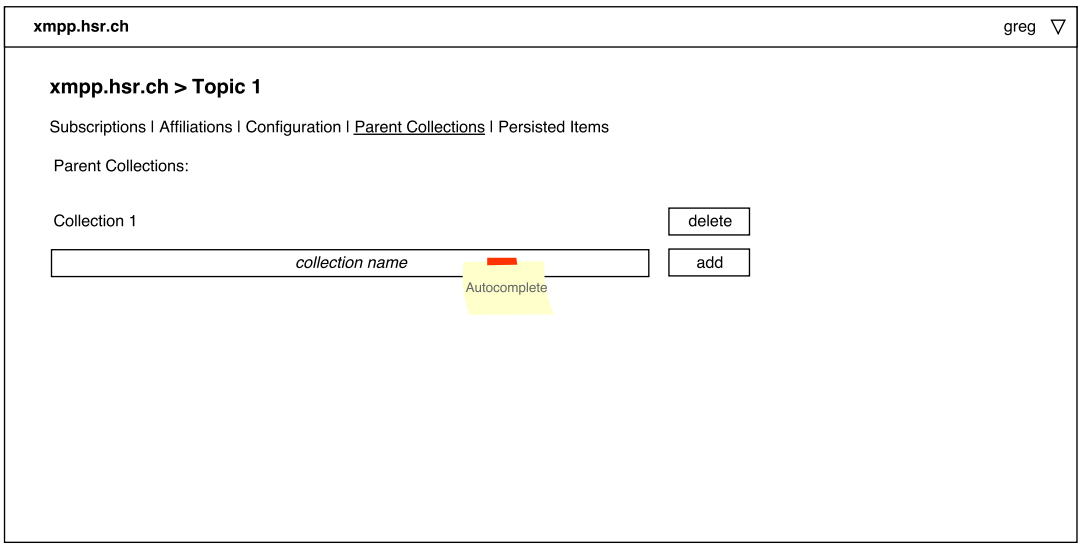


FIGURE 12: Topic parent collections items wireframe

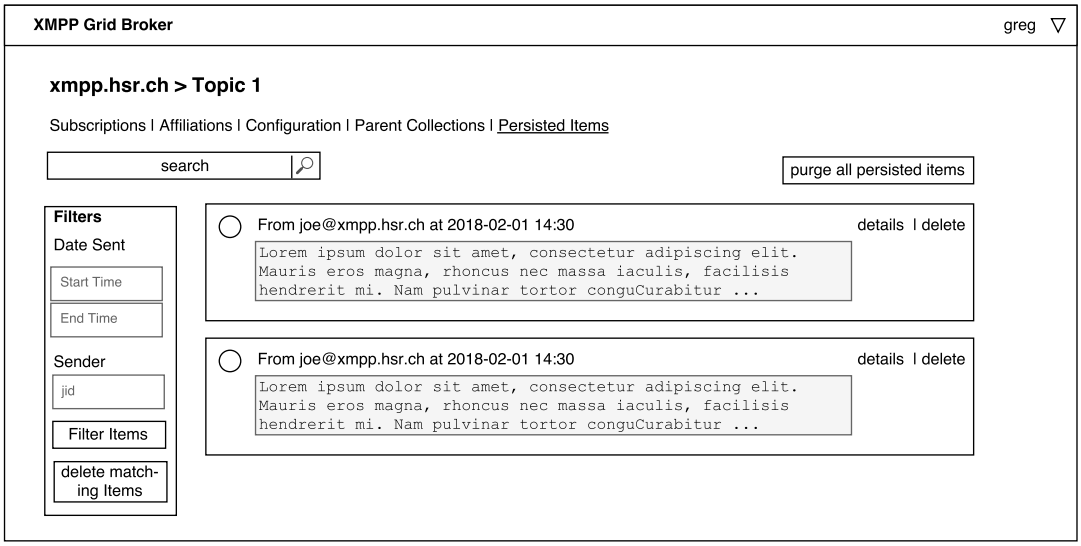


FIGURE 13: Persisted items wireframe

A.8 Comparison of XMPP Server and Libraries

A.8.1 Server

To find a suited server to develop, test and document our application with, we considered the three major open source *XMPP* servers, which are still under active maintenance and provide extensive documentation on their XEP-implementations.

For our implementation, we might require the following XEPs respectively RFCs. Any deviations are noted under "XEP/RFC Support".

XEP-0004 Data Forms

XEP-0030 Service Discovery

XEP-0059 Result Set Management

XEP-0060 Publish-Subscribe

XEP-0114 Jabber Component Protocol

XEP-0133 Service Administration

XEP-0178 Best Practices for Use of SASL EXTERNAL with Certificates

XEP-0206 XMPP Over BOSH

XEP-0248 PubSub Collection Nodes

RFC-7395 An XMPP Subprotocol for WebSocket

A.8.1.1 Openfire

Programming Language	Java	
Plugin Architecture	Java JAR[5]	
XEP/RFC Support	XEP-0133	Partial, also not explicitly supported[6]
	XEP-0178	Partial, also not explicitly supported[7]
	XEP-0248	Partial, as part of outdated XEP-0060[8]
	All other required XEPs are supported[9]	

[5] http://download.igniterealtime.org/openfire/docs/latest/documentation/plugin-dev-guide.html

[6] https://issues.igniterealtime.org/browse/OF-284

[7] https://github.com/Connectify/Openfire/blob/master/src/java/org/jivesoftware/openfire/net/SASLAuthentication.java and https://issues.igniterealtime.org/browse/OF-1191

[8] https://igniterealtime.jiveon.com/thread/38929

[9] http://download.igniterealtime.org/openfire/docs/latest/documentation/protocol-support.html

A.8.1.2 Prosody

Programming Language	Lua	
Plugin Architecture	Luascript[10]	
XEP/RFC Support	XEP-0059	Not supported
	XEP-0178	Not supported
	XEP-0248	Not supported
	All other required XEPs are supported[11]	

A.8.1.3 Ejabberd

Programming Language	Erlang	
Plugin Architecture	Erlang/Elixir[12]	
XEP/RFC Support	XEP-0178	Partial, commercially only[13]
	RFC-7395	Partial, not explicitly supported[14]
	All other required XEPs are supported[15]	

A.8.2 Libraries

To find a suited library to implement our application with, we considered various open source *XMPP* libraries, which are still under active maintenance and provide extensive documentation on their XEP-implementations. We also limited the programming languages to Python, Java and JavaScript as discussed in the project meeting of 2018-03-05 (see *A.5 Meeting Minutes*).

For our implementation, we might require the following XEPs. Any deviations are noted under "Limitations".

XEP-0004 Data Forms

XEP-0030 Service Discovery

XEP-0059 Result Set Management

XEP-0060 Publish-Subscribe

XEP-0114 Jabber Component Protocol

XEP-0133 Service Administration

XEP-0178 Best Practices for Use of *SASL EXTERNAL* with Certificates

XEP-0206 *XMPP* Over *BOSH*

XEP-0248 PubSub Collection Nodes

[10] https://prosody.im/doc/developers/modules
[11] https://prosody.im/doc/modules and https://prosody.im/doc/xeplist
[12] https://docs.ejabberd.im/developer/extending-ejabberd/modules/
[13] Server to server only in community edition
[14] https://docs.ejabberd.im/xmpp
[15] http://www.ejabberd.im/protocols

RFC-7395 An *XMPP* Subprotocol for WebSocket (mentioned only if differs from XEP-0206 implementation status)

Name	Language	Plugins	Limitations
SleekXMPP[16]	Python 2	Yes[17]	**Not Supported:** XEP-114, XEP-133, XEP-248, XEP-0206 **Partial:** XEP-0060[18]
SliXMPP[19]	Python 3	Yes[20]	**Not Supported:** XEP-114, XEP-133, XEP-248, XEP-0206 **Partial:** XEP-0060[21]
aioxmpp[22]	Python 3.4	Yes[23]	**Not Supported:** XEP-0114, XEP-0133, XEP-0178, XEP-248, XEP-0206
Smack[24]	Java	Yes[25]	**Not Supported:** XEP-0114, XEP-0206
Babbler[26]	Java	Yes[27]	**Not Supported:** XEP-0133, XEP-0248
XMPP-FTW[28]	JS(Browser)	Yes[29]	**Not Supported:** XEP-0114, XEP-0133, XEP-0248 **Unclear:** XEP-0206, XEP-0178 **Note:** Requires Server Abstraction
Stanza.io[30]	JS(Browser)	Yes[31]	**Not Supported:** XEP-0114, XEP-0133, XEP-0248 **Partial:** XEP-0178[32]
strophe.js[33]	JS(Browser)	Yes[34]	**Not Supported:** XEP-0114, XEP-0133, XEP-0248 **Partial:** XEP-0178[35]

[16] http://sleekxmpp.com/xeps.html
[17] http://sleekxmpp.com/create_plugin.html
[18] Client-side only
[19] https://github.com/poezio/slixmpp/blob/master/docs/xeps.rst
[20] https://github.com/poezio/slixmpp/blob/master/docs/create_plugin.rst
[21] Client-side only
[22] https://docs.zombofant.net/aioxmpp/devel/#from-xmpp-extension-proposals-xeps
[23] https://docs.zombofant.net/aioxmpp/devel/api/public/index.html#apis-mainly-relevant-for-extension-developers
[24] https://download.igniterealtime.org/smack/docs/latest/documentation/extensions/index.html
[25] https://github.com/igniterealtime/Smack/tree/master/documentation
[26] https://sco0ter.bitbucket.io/babbler/xeps.html
[27] https://sco0ter.bitbucket.io/babbler/customextensions.html
[28] http://docs.xmpp-ftw.org/manual/
[29] http://docs.xmpp-ftw.org/
[30] https://github.com/legastero/stanza.io/blob/master/docs/Supported_XEPs.md
[31] https://github.com/legastero/stanza.io/blob/master/docs/Create_Plugin.md
[32] Not explicitly supported
[33] https://github.com/strophe/strophejs-plugins
[34] https://github.com/strophe/strophejs-plugins
[35] Not explicitly supported

A.9 Personal Reports

A.9.1 Raphael Zimmermann

This project was an exciting journey for me because I wasn't familiar with the subject matter and the XMPP protocol at all.

I was surprised that many of the XEPs we used were still in a draft state even though they were around for over ten years and implemented in most servers. It was even more surprising to me that these XEPs were modified significantly, contradicting my wishful thinking to rely on the specification entirely. I also learned, that having too many optional features in a standard makes working with it tedious, especially if this functionality seems pivotal.

Because I had no prior experience with Angular, it took much work to understand all relevant underlying concepts. In hindsight, I must admit that I underestimated the complexity and familiarisation period. As a result, I was shifting my focus unintentionally from other practices that I usually focus on, such as clean layering. Nonetheless, I am glad to get familiar with Angular and I mostly enjoyed working with it.

Fabian and I are a well-practised team and as in our study project, working together was a pleasure.

A.9.2 Fabian Hauser

> *"The important thing is not to stop questioning."*
>
> — Albert Einstein

I find this quote very fitting for our thesis - not only was it our task to decide on options, but also keep questioning them to find the best possible solution.

We used architectural design decisions to find and question possible solutions. This technique supported our problem solving process greatly in my opinion. The most surprising turn resulting from an architectural decision during the project was our decision to write a client only application, which I wouldn't have expected from the task description.

During the project, I often had the feeling that we didn't advance as fast as I had hoped for. I think the main reason for this feeling is the time it took to get familiar with the complex XMPP standards.

Working together with Raphael was a very pleasant experience. Although we often worked remote from home, we had great discussions and conversations. Nevertheless, I think that it was helpful that we took the time to meet at least one time a week, which improved our communication and team spirit.

Declaration of Authorship

We, Fabian HAUSER and Raphael ZIMMERMANN, declare that this thesis and the work presented in it are our own, original work. All the sources we consulted and cited are clearly attributed. We have acknowledged all main sources of help.

Fabian Hauser

Raphael Zimmermann

Rapperswil, June 13, 2018