Contents

1 Foreword

I would like to thank Hannes Leutloff, who has spent countless hours on this project voluntarily and without whose tireless work on the user interface, the build infrastructure, mentoring and code reviews, this project would not have been possible.

2 Introduction

2.1 Terminology

data subject

data client

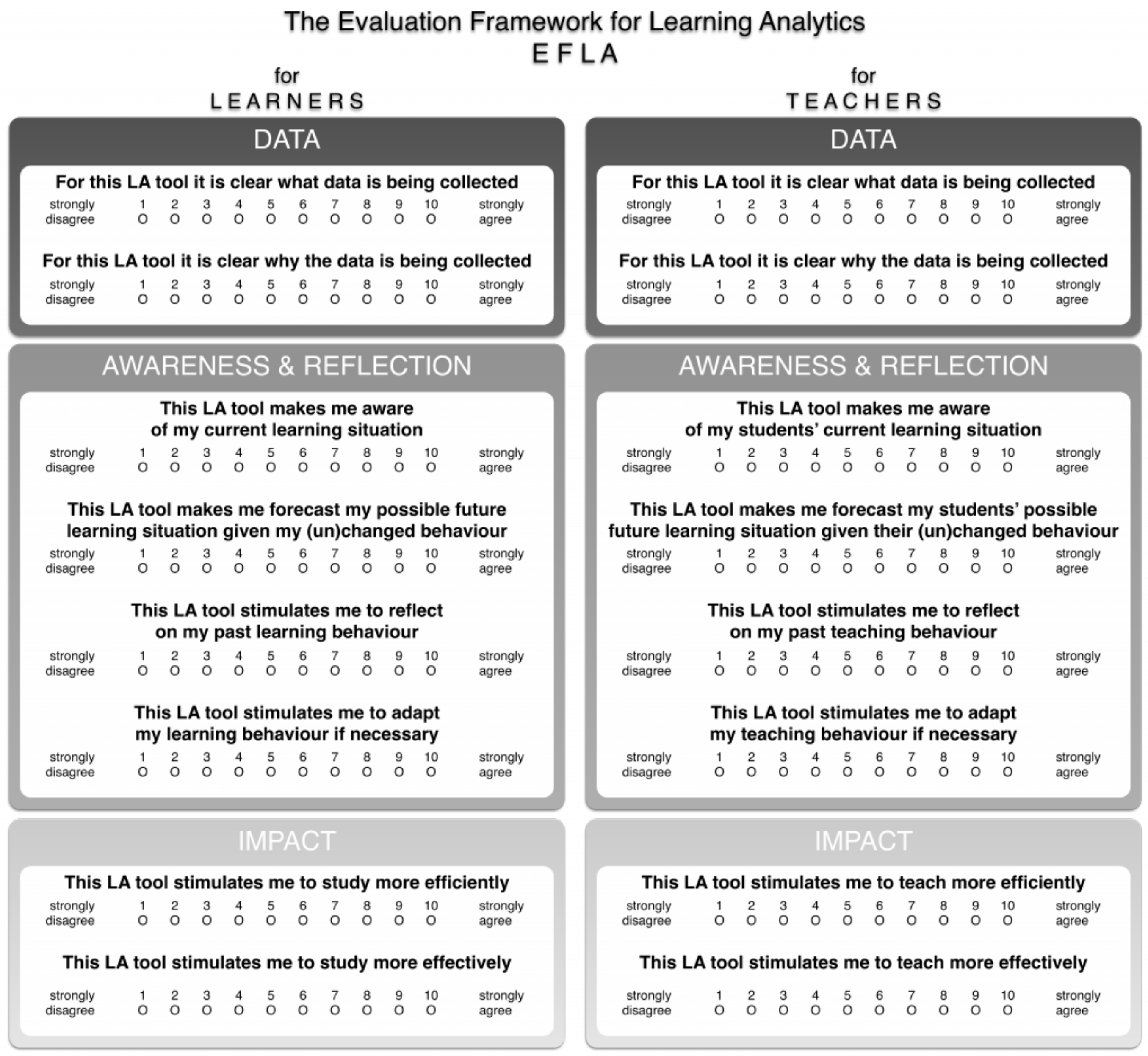
survey item

2.2 Previous Work

As part of the computational humanities seminar, which was held by Prof. H. Drachsler in winter 2017/18, Hannes Leutloff and I were tasked to digitize the evaluation framework for learning analytics (EFLA) (3) and to develop an online platform where the survey could be taken.

Feature	Description

Figure 2: Features of version 1.0.0 of the survey tool



The Evaluation Framework for Learning Analytics

EFLA

for LEARNERS

DATA

For this LA tool it is clear what data is being collected

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

For this LA tool it is clear why the data is being collected

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

AWARENESS & REFLECTION

This LA tool makes me aware of my current learning situation

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

This LA tool makes me forecast my possible future learning situation given my (un)changed behaviour

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

This LA tool stimulates me to reflect on my past learning behaviour

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

This LA tool stimulates me to adapt my learning behaviour if necessary

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

IMPACT

This LA tool stimulates me to study more efficiently

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

This LA tool stimulates me to study more effectively

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

for TEACHERS

DATA

For this LA tool it is clear what data is being collected

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

For this LA tool it is clear why the data is being collected

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

AWARENESS & REFLECTION

This LA tool makes me aware of my students' current learning situation

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

This LA tool makes me forecast my students' possible future learning situation given their (un)changed behaviour

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

This LA tool stimulates me to reflect on my past teaching behaviour

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

This LA tool stimulates me to adapt my teaching behaviour if necessary

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

IMPACT

This LA tool stimulates me to teach more efficiently

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

This LA tool stimulates me to teach more effectively

strongly disagree	1	2	3	4	5	6	7	8	9	10	strongly agree
	O	O	O	O	O	O	O	O	O	O	

Figure 1: Template for the EFLA survey (4)

The result was an online survey platform, where surveys similar to the EFLA can be created and hosted. While it was possible to create arbitrary survey items, some restrictions specific to the EFLA use-case applied (for a full list of features see table 2).

The original version of the survey tool was written in python and javascript for the server and user interface respectively. To be able to re-use code, the choice of language did not change with the new version.

2.2.1 Data Model

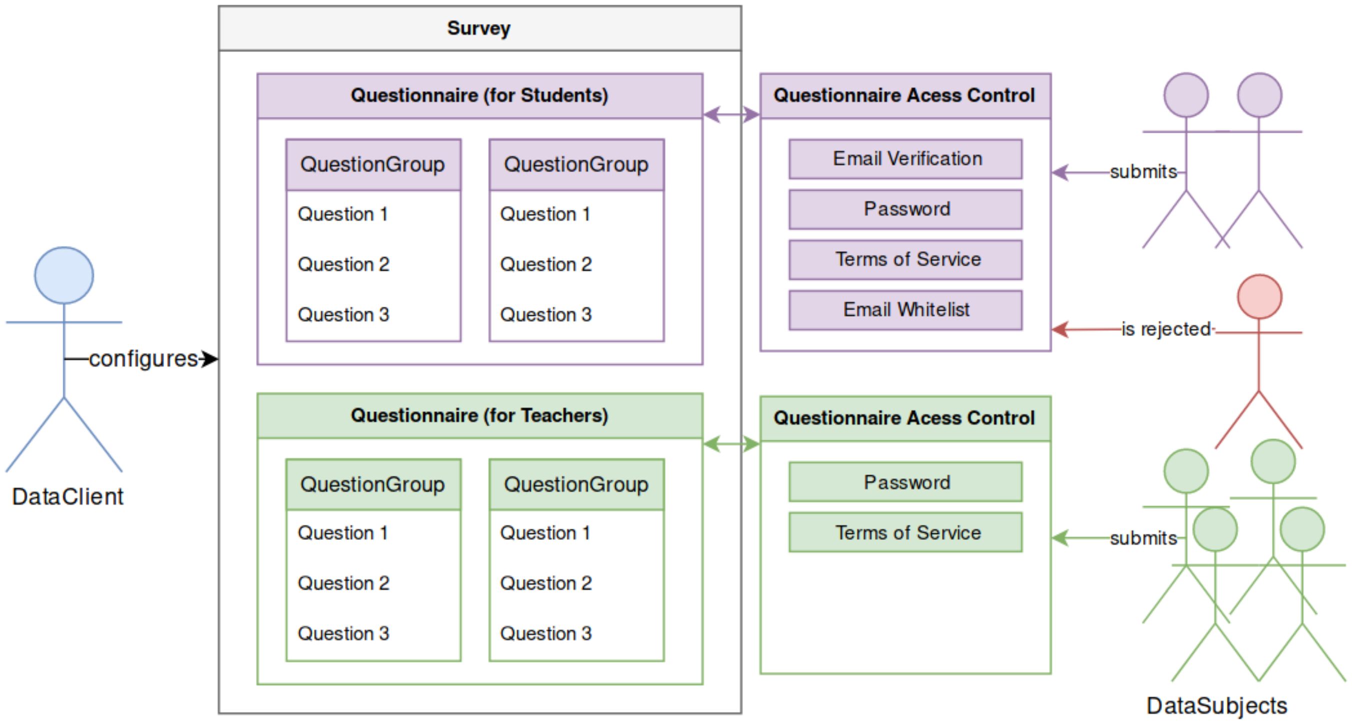
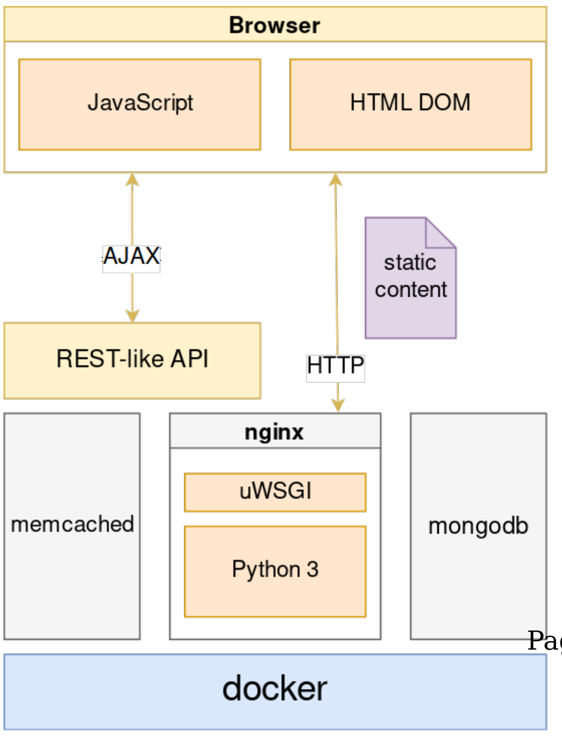


Figure 3: Data model of version 1.0.0

The data model for version 1.0.0 is closely coupled to the specific needs of the EFLA survey, where a single survey consists of one or more questionnaires, targeting multiple audience groups. Each questionnaire controlls DataSubjects' rights to submit by it's own set of access control modules, so that audience groups can be discriminated. Each questionnaire contains one or more question groups, which contains of one or more questions. All questionnaires within a survey would be published and retracted at once.

2.2.2 Architecture

The architecture for version 1.0.0 follows a simple client-server paradigm, where a web browser takes the role of the client. The server software consists of an application server, responding to API requests, a database and an in-memory key-value store for storing session data. There are two different user interfaces, one for data clients and one for data subjects. The data subjects' user interface is the page, where the survey is filled out and submitted. This page is rendered on the server using a templating engine and sent to the browser as a static HTML document. The user interface for data clients presents an editor, where survey content and settings can be created and modified. This user interface is realised as a one-page javascript application and was developed by



Hannes Leutloff. Deployment is handled through Docker, with the help of the docker-compose tool. The different parts of the server bundled up as separate docker images, which may be deployed on a docker host using docker-compose.

2.3 Goals

Describe goal of the thesis here -> extend software -> integrate with existing platforms

2.4 Use Cases

2.4.1 Motivated Strategies for Learning Questionnaire and Others

The Motivated Strategies for Learning Questionnaire is a psychological survey for accessing learning strategies. -> Why is it interesting? -> Write something about the self-regulation dashboard -> Anything I can cite on this? It is not possible to create the MSLQ with version 1.0.0 of the survey tool, as it only supports answers on a scale from one to ten, whereas the MSLQ uses a scale from one to five. -> Really? Also, for some other surveys which are of interest -> find examples <-, the order in which questions are presented to the data subject has to be randomized each time the survey is taken.

2.4.2 Embedding in Moodle

Moodle is an LMS used at Goethe University. The platform supports embedding of external tools inside of a course context. Because previous and simultaneous efforts by Prof. Drachsler's work group and collaborators already target Moodle, the new survey tool should also integrate with it. Integrating with Moodle also allows the survey tool to leverage Moodle's integration with the universities central authentication service to perform single sign-on.

2.4.3 Stand-Alone Survey Platform

In addition to directly embedding the survey tool into an LMS, it should be possible for a data subject to participate in a survey, if no LMS is used. This use case was carried over from version 1.0.0.

2.4.4 Data Provider as Part of the TLA Infrastructure

Prof. Drachsler's work groups' current efforts are targeted towards the implementation of a trusted learning analytics (TLA) infrastructure. This infrastructure will provide big data storage and analysis while complying with the european general data protection regulation (GDPR). The new survey platform should publish collected data to the TLA facts engine. Data from the survey tool may then be corellated with data collected from other sources to form a more complete picture on the data subject's learning behaviour.

2.5 Scope

In this thesis, the overall structure of the survey platform and the server-side software stack, which was designed and developed by Noah Hummel is discussed. Implementational details are discussed, if they are central to the workings of the software, or solve a particular conceptual challenge.

The user interface was designed and mainly developed by Hannes Leutloff, and only portions contributed by Noah Hummel will be discussed.

2.6 Methods used

Although this text follows a fairly linear structure, development focused mainly on rapid prototyping and did not follow the waterfall model of software engineering. Requirements were prioritized based on risk, meaning that more complex requirements and requirements for which the concept was not initially clear were implemented first. For these more risky requirements, development largely concentrated on creating a proof of concept first and then incrementally improving the prototype based on shortcomings of the previous iteration. Although the requirements did not change much during development, concepts changed constantly, which makes it hard to clearly separate concept from implementation. At some points, implementational details have to be discussed in the concepts section, as a previous iteration of the implementation informed the final concept.

3 Requirements Analysis

Requirements analysis was performed on the basis of the previously collected use cases. Required features and parts of the existing software that had to be refactored were identified.

3.1 Refactoring & Restructuring

3.1.1 Database Abstraction

The previous version of the survey tool uses a self-authored database abstraction library, the object-document-mapper (ODM). This library is capable of persisting python objects and relationships between them. It does this by serializing objects to JSON and storing using mongodb. Runtime interactions with these objects are intercepted and converted into database queries using the descriptor pattern (https://docs.python.org/3.7/howto/descriptor.html). This approach worked well for the limited use case, but it proved difficult to implement a transaction model that follows ACID properties.

To achieve transaction safety without compromising API transparency, an existing object-relational-mapper (ORM) was chosen to replace the ODM. Key factors for this choice were:

Maturity of the project. Data integrity is critical to the application. There is an increasing chance of bugs being found and resolved with increasing project age.

Support for inheritance mapping. Not being able to effectively use class hierarchies had proven itself to be a major burden on development speed during development of version 1.0.0, as attributes of related classes had to be explicitely duplicated. This was not only confusing to read but also difficult to maintain.

Support for JSON or hash map columns. JSON columns provide a convenient way to store multiple translations of a text column, without producing additional joins between the main table and supplemental translation tables. Storing multiple representations of a string inside a relational database usually involves a 1 to n relationship between the internationalised entity and it's translations. With the use of JSON columns, multiple translations of a string can be stored as a dictionary inside a single column of the entity itself.

In table 5 you can see a comparison between the some of the most popular ORM libraries available for Python 3. Because of the project age, previous experience with the library and it's reliability in production environments and the large feature set, SQLAlchemy was chosen to replace the ODM.

Feature \ ORM	SQLAlchemy	PeeWee	PonyORM	SQLObject
Declarative API	Yes	Yes	Yes	Yes
Eager loading of relationships	Configurable	No	Configurable	No
Lazy loading of relationships	Configurable	Yes	Configurable	Yes
Query caching	Yes	Yes	Yes	Yes
Cascades	update, delete	update, delete	delete	delete
Inheritance mapping	Yes	No	Yes	No
JSON columns	Using PostgreSQL	No	Yes	Using PostgreSQL
First released	2005	2010	2014	2003

Figure 5: Comparison of python ORM libraries

3.1.2 User Interface

As a parallel but somewhat separate effort, a major rewrite of the user interface was done by Hannes Leutloff. The user interface was re-written using VueJS. This was not explicitely required, but made integrating new features into the user interface easy.

3.2 Template Management

It became clear during negotiation with the stakeholder groups, that the set of questionnaires which are of interest are changing over time. As new research is published, and questionnaires become better statistically validated, the number of survey items needed to access a certain reasearch question ususally decreases. For these reasons, more current questionnaires can be preferred over older ones and it should be easy to integrate new survey content in the future. At the same time, once a questionnaire is reasonably well verified, multiple users may be interested in using the same questionnaire for different groups of participants. Version 1.0.0 had rudimentary template support, but templates are supplied as static YAML files on build and can not be changed or updated from the user interface. To overcome this limitation, the following requirements were established:

Questionnaires should be sharable between users. Once a questionnaire has been created on the platform, there should be a way to make it accessible to other users. This way, only one user has to enter the data.

Individual survey items should be sharable. To create slightly modified versions of a questionnaire, it is useful to re-use existing survey items.

Changes should be traceable. When survey items are shared with other users, and the creator of the item makes changes to the item, other users who use this item should be able to trace to change and the point in time when it occurred back to the creator. -> This lead to the tracker concept

3.3 Support for xAPI

Version 1.0.0 only supports limited data analysis capabilities, such as export of all response data for a given questionnaire as CSV and a rudimentary box plot visualisation. The new version should interface directly with the TLA infrastructure to allow for further data processing by TLA's analysis engine. TLA uses the xAPI statement format as it's common data representation. To provide data using xAPI to TLA, the following requirements were extracted:

Survey results should be published to an LRS via xAPI. A suiting xAPI representation of survey results has to be defined and should be transmitted to the LRS vie HTTP as defined by the xAPI specification (2).

The destination LRS should be configurable. This allows the location of TLA to change in the future. It also decouples the survey tool from TLA and allows interoperability with other LRSs.

xAPI statements must not be lost due to failure. When publishing data to an external storage, data integrity is no longer just determined by database integrity. Appropriate measures must be taken to ensure re-transmission in case of network failure. If transmission is not possible due to misconfiguration, an offline fallback method has to be provided.

3.4 Embedding & LTI Launch

To achieve compatibility between Moodle and the survey tool, the following requirements were established:

The survey tool has to implement LTI. LTI is used to launch external tools by Moodle. The survey tool has to recognize LTI launch requests and respond with an embeddable version of the survey.

Data subjects should not have to authorize. When using the survey tool from within Moodle, data subjects have already authorized with CAS in order to log in the LMS. Requiring another form of user authentication after this point would break seamless user experience.

LTI uses OAuth 1 to sign requests. This provides a way for the tool provider to verify the identity of the LMS and the authenticity of the LTI request. It was explicitely not required to implement OAuth as part of this work, but the design should take into account that future integration of OAuth should be easy.

3.5 Privacy Considerations

Since 25 May 2018, the general data protection regulation applies to members of the EU (1). The author is not in any way qualified to provide legal interpretation on the regulation. However, in consultation with Prof. H. Drachsler the following requirements were identified to conform with the regulation:

Data deletion for data subjects. Some personal data has to be stored on the server in order to identify data subjects between requests. Because of the right to be forgotten, there has to be a way to delete personal data stored for a specific data subject. It is sufficient if this is not automated and can be performed by the site's administrator.

No user accounts are available to external data clients. If third parties are allowed to use the service, we can not guarantee that they will follow GDPR guidelines. To avoid responsibility for third parties' actions on the site, there will be no way to sign up as a data client that does not involve contacting the administrator.

Other rights covered by the GDPR include the right to view and export personal data. Since all personal data collected is published to the TLA infrastructure, this will be implemented as part of the TLA trust engine and is not part of the survey tool.

4 Concept

4.1 Architecture

The overall architecture is the same as for version 1.0.0. The survey tool uses a classical client-server approach, where the client is a javascript application running inside a web browser. This approach enables everyone with a modern web browser to use the platform without having to install any additional software locally. The server-side software stack is deployed by using docker and exposed through a containerized web server. Communication between different containers on the server takes place on a virtual network which is not exposed to the internet. Communication between client and server is handled by a RESTful API which is provided by the server.

4.2 Workflows

In this section, intended workflows are detailed in the form of user stories.

4.2.1 Creating a Survey

> As a *data client* when I want to *create a survey* I navigate to the survey tool's *login page* in order to log into my account.
>
> I then *enter my credentials* and *confirm*.
>
> I am then presented with *a dashboard*.

4.2.2 Modifying and Translating a Survey

4.2.3 Participating in a Survey

4.2.4 Participating in a Survey through an LMS

4.2.5 Evaluating a Survey

4.3 Survey Content

During refactoring, the top-level organisational unit "survey" was removed and replaced by "questionnaire", as grouping multiple questionnaires inside a single survey only makes sense in a small set of use cases. For other use cases, this has proven to be confusing to users. Most of the time, a survey contains only a single questionnaire. Use cases, where grouping of multiple questionnaires into a single survey is appropriate can still be achieved by creating multiple questionnaires without any explicit grouping. The overall survey structure did not change significantly between versions 1.0.0 and 2.0.0 or above.

UML LIKE THINGY HERE

4.4 Ownership & Parties

To control API access to survey items, survey items had a single owner in versions 1.0.0. Owner refers to a data client who has access to the resource. In version superceding 1.0.0, the ownership model was expanded to allow n to n relationships between owning parties and owned resources. Ownership is also no longer restricted to data clients. These changes became necessary for two reasons. Survey item are no longer the only resource with access restrictions, as will become clear in section **??** "Mutation tracking". While survey items ususally only have a single owner - the author - tracking information has to be accessible to all parties sharing read access to the tracked resource. Because of the templating system, data clients may have read access to survey items of which tey’re not the author (templates). This requires a single resource to have multiple owners. Of course, a party may own more than a single resource, hence the n to n cardinality of this relationship. The reason for expanding ownership to data subjects is that the mechanism makes it easy to identify personal data. Data subjects by default own all resources which contain personal data. The “right to be forgotten” may then be implemented by simpy retrieving all owned objects for a given data subject and deleting them from the database.

4.5 Template Management

this whole section sucks ass -> In the following section the word “templates” is used as a substitute for survey items which are shared across multiple data clients. The word “copies” is used to describe survey items which are copies of a template.

The model suitable for shared content depends on the granularity of access control and modifiability needed. These two criteria present themselves as separate dimensions along which additional features have to be introduced to satisfy increasing demands of granularity.

The simplest variant of modifiability is not allowing data clients to modify templates at all. This was implemented in version 1.0.0 of the survey tool and can be achieved by using dedicated logic to instantiate survey items from static templates, i.e. files on the server’s hard drive. Subsequent modifications to the survey item instance do not affect the template and thus do not require special handling. To allow modifications on templates, instances of survey items in the database should be used as templates and not static files. Once data clients’ modifications affect the template, the challenge of synchronisation between the template and it’s copies arises. One potential solution to this challenge is to implement copies of templates by physically copying the template’s contents when instantiating a copy and then propagate subsequent modifications of the template to all of it’s copies. This solution’s time complexity scales linearly with the number of copies which exist for a given template, as all copies have to be updated in the database. It also scales linearly with regards to storage space, as each copy duplicates the template’s contents verbatim. The approach chosen for the survey tool uses references instead of verbatim copies. This reduces asymptotical time complexity for modification to constant time, as only the template has to be modified. It also signinficantly reduces used storage space. It does however also increases static overhead, since at least two block accesses will be needed to access a copy’s data.

Along the dimension of granularity of access control, <-

4.6 Integration with xAPI

4.6.1 Introduction to xAPI

xAPI, formerly known as TinCanAPI, is a data exchange standard closely resembling activity streams (https://www.w3.org/TR/activitystreams-core/). It was developed by the Advanced Distributed Learning (ADL) Intiative as a successor to SCORM and allows the exchange of experiential information in the form of statements. These statements use JSON as their data format and include a minimum of three semantical objects, “actor”, “verb” and “object”. The data is transmitted using HTTP, following REST principles.

```
{
    "actor": {...},
    "context": {
        "contextActivities": {
            "grouping": [
                {...}
            ],
            "parent": [
                {...}
            ]
        },
        "extensions": {
            "http://activitystrea.ms/schema/1.0/place": {...}
        },
        "language": "en",
        "platform": "st3k101 via localhost"
    },
    "id": "896ef7f1-8d5c-4729-b028-e9d72df47fe8",
    "object": {...},
    "result": [
        {...}
    ],
    "timestamp": "2018-09-27T14:32:15.009513",
    "verb": {...}
}
```

Figure 6: Anatomy of an xAPI statement

Every xAPI statement may also include a “timestamp” stating the issue date and time of the statement, a “context”, providing additional information about the event or a “result”, detailing the outcome of the event. The format is also extensible, additional information can be provided in the “extension” object.

4.6.2 xAPI Statement Design

There are several actions which will trigger sending of an xAPI statement:

1) A data client logs in into the survey platform.
2) data subject launches an embedded survey.
3) data subject answers a single question.
4) data subject answers a known survey item (whole questionnaire or dimension)
5) data client updated the xAPI activity ID of a survey item.

Figure 7: List of cases where xAPI statements are emitted

For each of these cases, an xAPI statement had to be designed. All of these statements share at least some information. The `context` object of all xAPI statements emitted by the survey tool includes the `platform` and `language` and `extensions` attributes. The `language` attribute always contains an RFC 5646 (https://tools.ietf.org/html/rfc5646) compliant representation of the language the action was performed in. The `platform` attribute always contains the string `st3k101`, identifying the origin of the statement. The `platform` attribute may also include the URL of the source LMS in the case LTI was used. In this case, the URL is prepended as `st3k101 via SOURCE_LMS_URL`. The `extensions` obejct of the `context` also includes a geolocation for the client in RFC 7946 compliant GeoJSON format (https://tools.ietf.org/html/rfc7946). Below is a concept for what additional data should be included in the statements listed in 7.

#	Actor	Verb	Object	Result	Context
1	data client	logged in	login page	-	-
2	data subject	accessed	questionnaire	-	-
3	data subject	answered	question	response value	parent dimension AND questionnaire
4	data subject	answered	qestionnare OR dimension	response value	parent item, if any
5	data client	updated	questionnaire OR dimension OR question	new acitivity ID	-

Table 1: Concept for information included in emitted xAPI statements

Objects are identified by their `objectType`, `type` and `id` in xAPI, whereas verbs are just identified by their `id`. For the purpose of the survey tool, all objects share the same `objectType`, the `Activity`. The verb's `id` is semantically equivalent to an object's `type`. It is common practice to use a URL as identifier or type, which will return a human readable description of the item via HTTP GET. In theory, there's no correct identifier to use when designing xAPI statements, as the standard does not prescribe the use of any specific verbs or objects. In practice, several registries with commonly used verbs and objects exist and should be consulted when choosing which identifier or type to use. This avoids re-definitions of already existing items and increases homogeneity among statements by different adopters of the standard. For this reason, the verbs and objects used in table 1 had to be translated into already existing verbs and object. The results are detailed in table 2. For some of the objects, no suitable definitions existed. For those objects, dummy identifiers or types were used, which follow the URL format, but use `http://fantasy.land/` as a prefix.

In order to corellate objects in emitted xAPI statements with survey items in the survey tool, all survey items have a user-modifiable xAPI acitivity ID associated with them. This identifier is used as

Verb	Identifier
logged in	`https://brindlewaye.com/xAPITerms/verbs/loggedin`
accessed	`https://w3id.org/xapi/dod-isd/verbs/accessed`
answered	`http://adlnet.gov/expapi/verbs/answered`
updated	`http://activitystrea.ms/schema/1.0/update`
Object	**Type**
login page	`http://activitystrea.ms/schema/1.0/page`
questionnaire	`http://id.tincanapi.com/activitytype/survey`
dimension	`http://fantasy.land/dimension`
question	`http://adlnet.gov/expapi/activities/question`

Table 2: Used xAPI verb identifiers and object types

```json
"object": {
    "definition": {
        "description": {
            "en-US": "This is a particular scale of a survey, it usually contains
                multiple questions."
        },
        "name": {
            "de": "5. Anstrengung"
        },
        "type": "http://fantasy.land/dimension"
    },
    "id": "<bla@blubl.net>:lernstrategien_wild_schiefele--5_anstrengung",
    "objectType": "Activity"
}
```

Figure 8: Example of how a dimension is represented as an xAPI acitivity object

the object's `id` in xAPI statements. These identifiers are not modifiable in copies of templates, which means that all instances of a copy will use the same xAPI activity ID. This is useful for conducting meta-analyses, where all results for a certain template could be included in the analysis, regardless of who conducted the survey. During testing, this presented itself as an issue, because results for the same template, which were collected by different data clients would not be distinguishable, as they all used the same identifier. For this reason, the email adress of the data client who conducts the survey is added as a prefix to all xAPI activity IDs before sending. -> Drawback: not normalized, potential fix: use Instructor <- In figure 8, an example of how survey items will be represented in xAPI is given.

Actors may be represented in three different ways using xAPI. The identifying feature is either the person's email adress in the case of the `mbox` and `mbox-sha1sum` actor types, an account id in combination with a URL where the account is located in the case of the `account` actor type or OpenID credentials in the case of the `openid` actor type. Data clients are represented as `mbox` actor types, while data subjects may be represented by as `mbox-sha1sum` actor types or, in the embedded use-case, as an `account` actor type using their LTI user identifier. The latter is necessary, because the email address might not be available though LTI. The LTI user ID is, contrary to prior suppositions, not useful for data analysis, as Moodle will use the user entities database ID. Moodle does however communicate a username via LTI, which for the specific Moodle instance at Goethe University is the same as the data subject's CAS ID. To retain compatibility with other LMSs while taking this finding into account, the username will take precedence over the LTI ID, if it's present in the LTI request. The LTI ID is also not globally unqiue, as there may be separate LMSs which use the same IDs for different users. For this reason, the LTI user ID is prefixed with the identifier of the source LMS, which is always present in LTI requests.

The recipient of the statements is determined by the object which is acted upon. This makes intuitive sense, as the person owning a certain survey item is the one interested in collecting data on it. All survey items are on the highest level organized into some questionnaire and there's no use-case for different data consumers for individual parts of the questionnaire. For this reason, recipients are configured on a per-questionnaire basis. For objects which don't have an owner, for example the survey tool's login page, a default recipient is configured system-wide.

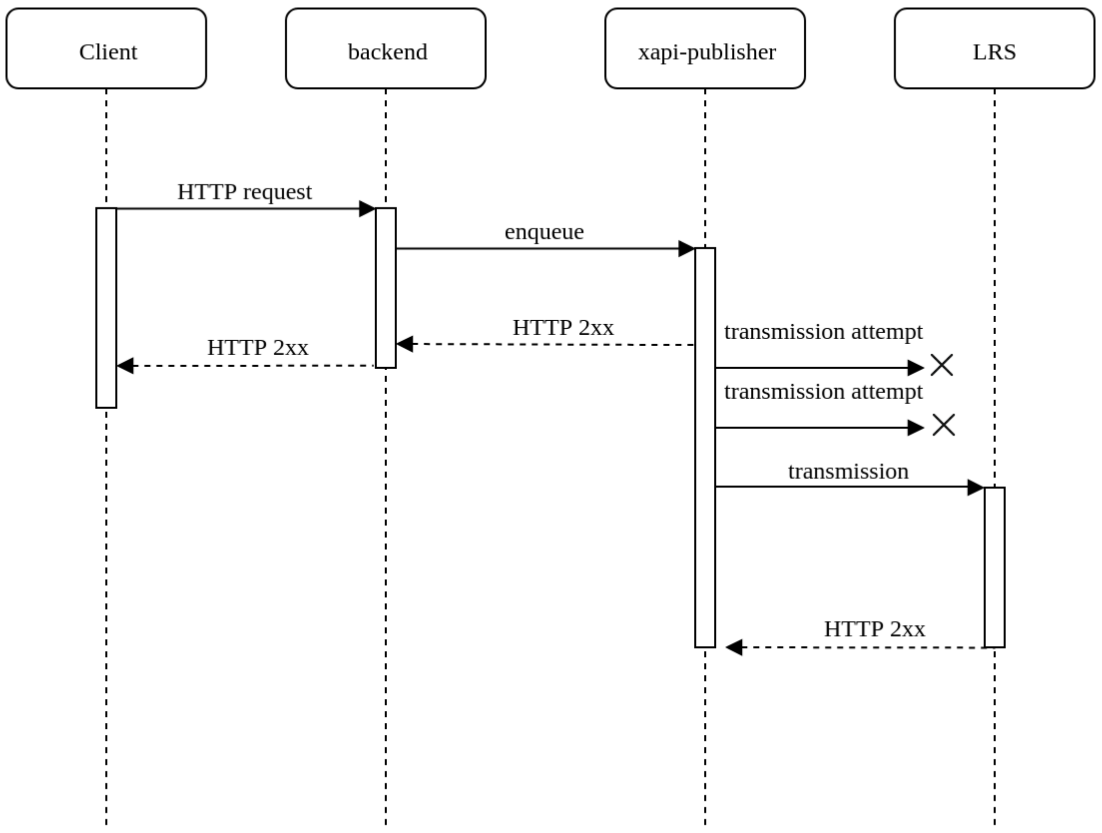


Figure 10: Asynchronous (non-blocking) transmission of xAPI statements

Sending of xAPI statements is separated from the API, as sending and possible re-transmissions should not block the API from responding to requests. This was discovered during testing, when a misconfigured xAPI recipient was used. HTTP timeouts are usually in order of seconds and unsuccessful connection attempts are retried. This resulted in the site becoming unresponsive when submitting the survey, as the submission of a survey would cause a large number of xAPI statements to be sent. Figures 9 and 10 illustrate this issue.

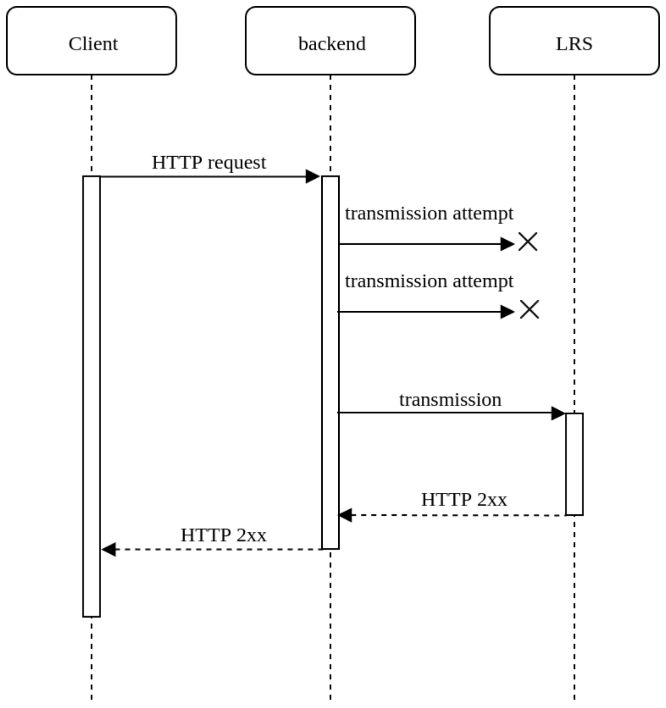


Figure 9: Synchronous (blocking) transmission of xAPI statements

4.7 Embeddedable User Interface

The embedded user interface is launched by the LMS using the LTI protocol. Information about the user source LMS are already present in the request body. Since there's some time between the LTI launch request and the survey submission, the request information needed to identify the user when submitting, for example their user ID or the source LMS, has to be stored on the server for this period of time. To achieve this, an LTI request is treated as a login action by the data subject and the required data is stored in the user's account. If a user accesses the service for the first time, a new account is provisioned for them. A valid LTI request is treated as sufficient authentication for the data subject, as the source LMS already authenticated the user prior to them accessing the survey. To identify the source LMS, a random token, the consumer key, is generated in the back-end for every questionnaire. To embed the survey within the course context, an LTI request with the correct combination of request URL and consumer key has to be made by the LMS.

4.8 Privacy Considerations

As mentioned in the section above, when a data subject participates in a survey, a user account is provisioned for them. Removal of this account and all associated personal data is performed via the API when authenticated as an admin user. In order for the admin to know which account to communicate to the API, the API allows admin users to query for existing user accounts. Data removal is limited to admin users, as removal of accounts by data subjects themselves would require some sort of authentication mechanism for data subjects. When an email address is present for the data subject, authorization via email is a possible solution for this. In this scenario, the data subject would receive an email with a one-time token, which can be used to remove their personal data. If there's no email present for the data subject, for example when the account was provisioned using LTI, or if the data subject hasn't got access to their email account, this mechanism fails. Hence, email as sole mechanism for data removal is not a viable option at the moment.

5 Implementation

5.1 Architecture

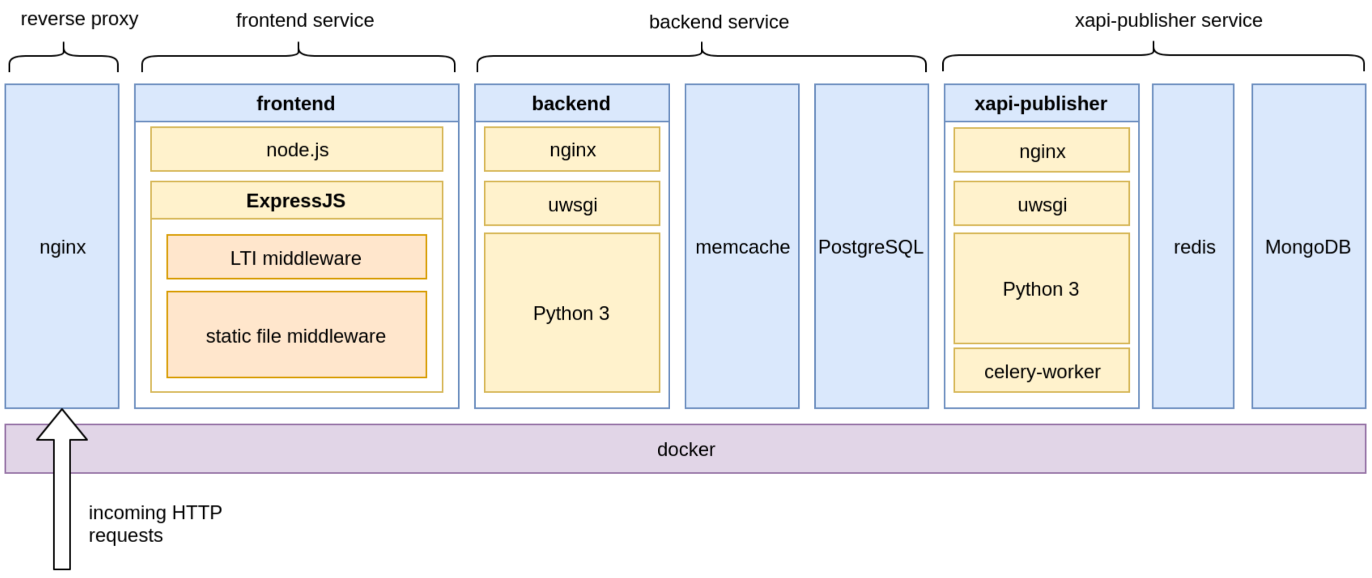


Figure 11: Server-side software stack

The server-side architecture is composed of multiple services. The "frontend" service is responsible for serving the user interface statically and intercepting LTI requests. The "backend" service is responsible for serving the API, user and session management and communication with the database. The "xapi-publisher" service is responsible for transmission of xAPI statements generated by the backend service. Each service uses slightly different technologies, depending on the requirements of the service. The API and "frontend" service are exposed through an nginx web server, which acts as a reverse proxy, forwarding incoming HTTP traffic to the appropriate docker containers.

5.1.1 The “backend” Service

The “backend” service is implemented using the “flask” library, which allows Python to interface with a web server using the web server gateway interface. Similar to PHP or CGI extensions, the nginx web server will accept incoming requests. It will then, using uwsgi, fork a python interpreter which will respond to the request. This means that no data can be persisted in the Python application itself, as the Python context will be discarded for every request. To persist data, a PostgreSQL database and a memcached in-memory key-value store are used. True persistent data, for example survey items, are stored in the database, whereas semi-persitent data like user session are stored in memcached. This design allows for horizontal scalability, as multiple instances of the “backend” containers can be used with the same database, as no data dependencies between the “backend” containers exists. The only bottleneck in this scenario is the shared database, which could be solved long-term by replacing the single PostgreSQL by a load balancing database cluster.

5.2 The “frontend” service

The “frontend” service consists of a simple node.js application running the ExpressJS web server. For the web server, two middlewares are provided, one for serving static files and another for intercepting LTI launches. This is necessary, since the user interface has to be configured for each LTI launch before serving it to the data client.

5.3 The “xapi-publisher” Service

The “xapi-publisher” container mimics the design of the “backend” container closely. In addition to flask and uwsgi, it also runs several worker threads, which are responsible for asynchronous sending of xAPI statements. Since xAPI statements are JSON documents, a document based database is used instead of a relational database to persist xAPI statements between requests. For inter-process communcation between web server and worker threads, a task queue consisting of the celery library and the redis in-memory database is used.

The functionality provided by the “xapi-provider” container is seperate from the “backend” container, duplicating most of the architecture already present. This might seem sub-optimal at first, as it violates the DRY (don’t repeat yourself) principle to the extent that configuration for two seperate containers has to be created and maintained. It does however allow for better seperation of concerns. The “backend” container already handles business logic and user management and there is a unique set of challenges that has to be solved for publishing xAPI statements, which would unnecessarily increase the complexity of the “backend” service. One such challenge is, that sometimes data needed to create an xAPI statement is present in a request before the data is actually valid an can be sent. For example, when a data subject answers a survey using the stand-alone survey interface, their answers are submitted to the server but only become valid, after they have verified their email address. The required data to build the xAPI statement, including the data subject’s IP adress, has to be stored on the server until this point. In order to avoid convoluting the “backend” service’s data model and business logic to account for this, a seperate service solely responsible for temporarily storing and safely transmitting the xAPI statement is used. This allows the “backend” service to only use minimal logic for communicating with “xapi-publisher” service. It also allows maintainers of the codebase to make changes to the publishing behaviour without having to know the internal workings of the “backend” service.

5.4 Data Model

5.4.1 Parties & Ownership of Resources

Figure 12: Database schema for parties & ownership

5.4.2 Survey Structure

Figure 13: Database schema for survey items

5.4.3 Template Management

Figure 14: Database schema for a single question including foreign key constraints

5.4.4 Internationalization

5.4.5 Mutation Tracking

Figure 15: Database schema for modification tracking

5.5 User Authentication

5.5.1 Bearer Token Authentication

5.5.2 LTI Tool Launch

5.6 RESTful API

schema validation dependency injection

5.7 Template Lifecycle Management

5.8 xAPI Publisher Microservice

5.9 User Interface

6 Setup & Configuration

6.1 Database Versioning & Migration

6.2 Dockerized Deployment

7 Summary & Outlook

7.1 Known Issues & Caveats

8 Abbreviations

API Application programming interface; a software interface for interoperability between applications.

CSV Comma separated values; a plain-text data exchange format for homogenous data.

ORM Object-relational-mapper; a software library for persisting objects in object-oriented languages in a relational database.

ODM Object-document-mapper; a software library for persisting objects in object-oriented languages in a document based database.

xAPI Experience API, formerly TinCan API; an API specification for exchanging data about learning activities.

REST Representational state transfer; a paradigm for API design.

JSON Javascript object notation; a plain-text data exchange format.

ACID Atomicity, consistency, isolation, durability; a set of properties for database transaction, aiming to guarantee data validity in the event of failure.

SQL Structured query language; a language for interfacing with database systems.

TLA Trusted learning analytics; a big data storage and analysis infrastructure developed at Prof. H. Drachsler's work group.

LRS Learning record store; a big data store for xAPI statements.

LTI Learning technologies interoperability; a protocol for integrating third-party services with an LMS.

LMS Learning management system; a content management system specifically designed for learning environments.

CAS Central authentication service; the user authentication service provided by Goethe University's HRZ.

HRZ Hochschulrechenzentrum; Goethe University's data center and IT service provider.

List of Figures

List of Tables

References

[1] The European Commission. Data protection in the EU. https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_en, 2016. Accessed: October 15, 2018.

[2] The Advanced Distributed Learning Initiative. xAPI specification. https://github.com/adlnet/xAPI-Spec/releases/tag/xAPI-1.0.3, 2016. Accessed: October 15, 2018.

[3] Maren Scheffel. The Evaluation Framework for Learning Analytics. http://hdl.handle.net/1820/8259, 2017. Accessed: October 15, 2018.

[4] Maren Scheffel. The Evaluation Framework for Learning Analytics, greyscale template. http://www.laceproject.eu/evaluation-framework-for-la/, 2017. Accessed: October 15, 2018.