

communicated by:
Lehr- und Forschungsgebiet Informatik 9
Prof Dr. Ulrik Schroeder



The present work was submitted to the Learning Technologies Research Group
Diese Arbeit wurde vorgelegt am Lehr- und Forschungsgebiet Lerntechnologien

Enhancing Usability and Data Integrity in LiMoxAPI for LA Developers

Das Verbessern der Benutzerfreundlichkeit und Datenintegrität von LiMoxAPI für LA Entwickler

Bachelor-Thesis

Bachelorarbeit

presented by / von

Sieler, Lea

434459

Examined by / Begutachtet von

Prof. Dr.-Ing. Ulrik Schroeder

Prof. Dr. Jan Borchers

Supervised by / Betreut von

Sergej Görzen, Sarah Sahabi

Aachen, February 22, 2025

Contents

List of Figures	iii
1 Introduction	2
1.1 Motivation	3
1.2 Goal	3
2 Scientific Background	4
2.1 Iterative Enhancement	4
2.2 Learning Analytics	4
2.2.1 Tasks and Purpose	4
2.2.2 LA in a XR Context	5
2.3 Experience API	6
2.4 UI and UX Design	7
3 Related Work	8
3.1 OmiLAXR Ecosystem	8
3.1.1 xAPI Definitions Registry	8
3.1.2 xAPI4Unity	9
3.1.3 The Researcher Companion Panel	9
3.1.4 OmiLAXR Framework	9
3.2 LiMoxAPI	10
3.2.1 Technical Stack	11
3.2.2 Frontend Components	11
4 First Requirements and User Evaluation	14
4.1 Reviving Requirements	14
4.2 Evaluation	15
4.2.1 The Setting	15
4.2.2 The Study	16
4.2.3 The Results	16
5 Implementation	19
5.1 Download and Reset Button	19
5.2 Helpful Explanations for an improved UX	20
5.3 Pulse Monitor	20
5.3.1 X-Axis Labeling	20
5.3.2 History Log	21
5.3.3 Additional Statement Type Graphs	21

5.4	Logging Console	22
5.5	Filter Enhancements	23
5.5.1	Simplified Filter Rule Definition	23
5.5.2	Analysis of Single Filters	23
5.5.3	RegEx	24
5.5.4	Filter Logic	25
6	Final Evaluation	28
6.1	The Evaluation	28
6.1.1	Pulse Monitor	28
6.1.2	Info Component	29
6.1.3	Filter Component	29
6.1.4	Logging Console	30
6.1.5	Button Improvements	30
6.2	Overall Results	30
7	Future Work	31
7.1	Enhancing Existing Features	31
7.2	Changes in LiMoxAPI's Architecture	32
7.2.1	Integration of the YetAnalytics SQL LRS	32
7.2.2	Developing a Plugin System	32
7.2.3	Creating a REST-API	33
8	Conclusion	34
	Appendix	35
A	Bibliography	36
B	User Evaluation Script	39
	Glossary	42
	Acronyms	43

List of Figures

2.1	LA life cycle adapted from [KE15].	5
2.2	Screenshot of a possible xAPI statement consisting of an actor, verb and object	7
3.1	Workflow of the OmiLAXR Ecosystem, taken from [Gö24]	9
3.2	Concept Architecture of LiMoxAPI adapted from [TSL24]	10
3.3	Internal Architecture of LiMoxAPI adapted from [TSL24]	12
3.4	Login-View of LiMoxAPI taken from [TSL24]	13
3.5	Dashboard-View of LiMoxAPI taken from [TSL24]	13
5.1	Example of the downloaded Login Credentials stored in JSON format . .	19
5.2	The new Reset Dialog Formulation	20
5.3	The History Log of the Pulse Monitor	21
5.4	The final Dashboard-View of LiMoxAPI	22
5.5	The Filter Settings Dialog with the added Dropdown Menu, RegEx Search and Filter Logic	23
5.6	The Filtered Statements Log of Single Filters	24
5.7	Add Filter Logic Dialog	25
5.8	Simple Filter Logic Example	27
7.1	Possible Docker Architecture	33

Abstract

When creating complex learning environments, most Learning Analytics (LA) developers have the same problems and limitations in their work:

The testing process of their learning platforms is tedious and time-consuming, due to complex Learning Record Store (LRS) infrastructures and connection issues, thus the verification of the generation of their learning data cannot be guaranteed [GHS24].

To mitigate these problems, the live monitoring tool LiMoxAPI [TSL24] (Live Monitoring of xAPI Statements) has been developed for VR developers which runs locally on the computer and periodically fetches incoming statements from a specified LRS.

It is a part of the OmiLAXR Ecosystem which is provided by the RWTH research group Learning Technologies [HEGS22].

LiMoxAPI initially formed a solid base for verifying the generation of learning data, however when taking a closer look, it was obvious that LiMoxAPI provided a lot of potential for further improvements.

The application was stiff, did not provide any interplay between the components and was vastly unintuitive for the user.

This thesis enhanced the usability and data integrity of LiMoxAPI and widened the accessibility of the application for all LA developers instead of just focusing on VR development.

This was achieved by first retrieving new requirements for LiMoxAPI by conducting a user evaluation with two participants from different LA application fields in the beginning of the project.

A second evaluation has been conducted at the end of the project to present the final results as well as to gain additional feedback.

The achieved results are an improvement of LiMoxAPI's functionality.

This was done by implementing features that lead to a better interplay between the components and providing enhanced options for the user to interact with the application.

A special focus lay on the filter component which now allows the user to perform more complex actions with their generated xAPI statements due to the provision of filter logic, the enablement of regular expressions (RegEx) and the option to reset and log the filtered statements individually for each filter.

Additionally, an improved user experience was achieved due to making the application more intuitive. Therefore, the different components have been restructured, additional explanatory texts have been added and proper feedback is provided to the user.

LiMoxAPI is now accessible to a wider range of users and allows an enhanced way to monitor and verify the generation of learning data.

Chapter 1 Introduction

Learning Analytics (LA) is a broad and interdisciplinary research field which enables the analysis and improvement of learning processes as well as the representation of learning data using learning environments [Sie12]. Researchers try to identify successful learning patterns as well as to detect misconceptions. Subsequent to that, they can introduce improvements for better learning progress [GTE⁺17].

This analyzed learning data is generated using learning platforms. But how can we verify the correctness of that data?

Learning data uses a specific syntax following the experience API (xAPI) specification. These xAPI Statements are stored in a Learning Record Store (LRS), waiting to be analyzed by researchers or educators [VRL15].

In order to test if the correct data was generated or to test if xAPI statements have been generated at all, LA developers usually have to keep a stable connection to a LRS and manually refresh the database in order to see if new statements have arrived. This has shown to be time-consuming and effortful [GHS24].

This is why LiMoxAPI has been introduced being a former Bachelor Thesis of G. Thiesen [TSL24]. It is a live monitoring tool which fetches learning data from a local LRS and monitors the fetched statements in a dashboard.

Originally, LiMoxAPI was developed for LA developers from a Virtual Reality (VR) domain and became a part of the OmiLAXR Ecosystem [GHS24] provided by the research group Learning Technologies [HEGS22].

The end result of the thesis was a solid prototype for VR developers, but it turned out to be quite stiff while lacking interaction options for the user and complex features.

How could the usability of LiMoxAPI be improved while making it accessible to a wider range of users without being limited to VR development? This is one of the main research questions of this thesis.

The thesis is structured as follows: After introducing the reader to the problem definition and motivation of this project, the required scientific background is provided, defining the research field of LA and the xAPI specification as well as looking at user interface and user experience design concepts.

Subsequent to that, related work of this thesis is presented, regarding the OmiLAXR Ecosystem and the initial development of LiMoxAPI. The next chapter intensively looks at the initial state of LiMoxAPI, defines first requirements for the enhancement of the application and presents a user evaluation that has been conducted.

Subsequently, the following chapter describes the implementation process of the ideas that were retrieved from the user evaluation in detail.

Then, a final user evaluation was held and summarized the opinions of the participants after seeing the final version of LiMoxAPI.

This paper ends with an outlook on possible future work and a final conclusion about the outcome of the project.

1.1 Motivation

LiMoxAPI has been developed to provide a proper live monitoring tool for VR developers. Back then, developers of VR learning environments had to keep a stable connection to Learning Locker (LL) when trying to test their environments.

Nevertheless, LL is a LRS which has a complex infrastructure thus a stable and continuous connection cannot be guaranteed. These complaints have been retrieved from a user study of the research group [GHS24].

Developers were in need of a simple and lightweight analysis tool for the verification of their learning data. With LiMoxAPI, a first prototype could be achieved.

LiMoxAPI provides a small dashboard with a pulse monitor which graphically displays the amount of incoming statements per fetch from the LRS, but it also has the feature to log the incoming statements and provides a counter for specified filters.

These components form a solid base, nonetheless, there is still a lot of room for improvement.

The single components are quite unintuitive and do not provide possibilities for a lot of interaction with the user. There is no interplay between the different features and everything is quite encapsulated.

Thus, LiMoxAPI is in need of a more intuitive user interface, more complex features as well as an improved interplay between the components in order to ensure an enhanced user experience and data integrity for LA developers.

1.2 Goal

LiMoxAPI provides a lot of room for improvement. Therefore, the aim of this project is to enhance the functionality of the application by implementing additional required features and by ensuring a better user experience for LA developers.

It needs a better interplay between its single components, such that developers have a greater chance to analyze and verify the generation of their learning data in form of xAPI statements.

Further scientific background will be given in the chapter Scientific Background.

Although originally being developed for VR developers, the application should be expanded and made accessible to a wider range of users such that LA developers from all fields are able to interact with it, without being limited to the use of the OmiLAXR Ecosystem.

The required features will be extracted via user evaluations with participants from different application fields of LA in order to gain a widened perspective about the analysis and usage of learning data.

Chapter 2 Scientific Background

This section specifies the scientific background of this thesis. It will describe the followed software development approach, the concept of Learning Analytics, the Experience API specification as well as Learning Record Stores.

Apart from that, user interface- and user experience- design goals and principles will be defined which have been followed while enhancing LiMoxAPI.

2.1 Iterative Enhancement

The software development approach Iterative Enhancement by Basil et al. [BT75] will serve as an orientation for the implementation process of the project.

Iterative Enhancement describes a top-down, stepwise refinement approach to software development [CV97] and starts with a simple implementation of a chosen sub-project.

The developer creates a project control list which contains all the tasks needed to achieve the final implementation.

In each iterative step, a new task is chosen from the list. After its completion, the task gets removed and the process of choosing a new task restarts. At any time, the developer can make extensions as well as design modifications in their implementation.

The core idea of this approach is to always have a running version of the project. There are iterative enhancements made on existing versions until the full system is implemented [BT75].

2.2 Learning Analytics

Siemens et al. [Sie12] describe LA as “the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs”.

In order to understand what this means in detail, the following subsections provide information about the task and purpose of LA as well as looking at a specific LA application field which led to the development of the OmiLAXR Ecosystem [HEGS22].

2.2.1 Tasks and Purpose

The term LA is lacking a universal definition but generally refers to the process of extracting valuable insights from collected learning data and developing corresponding methods and tools to achieve an enhancement in the learning experience [Clo13, CDST12].

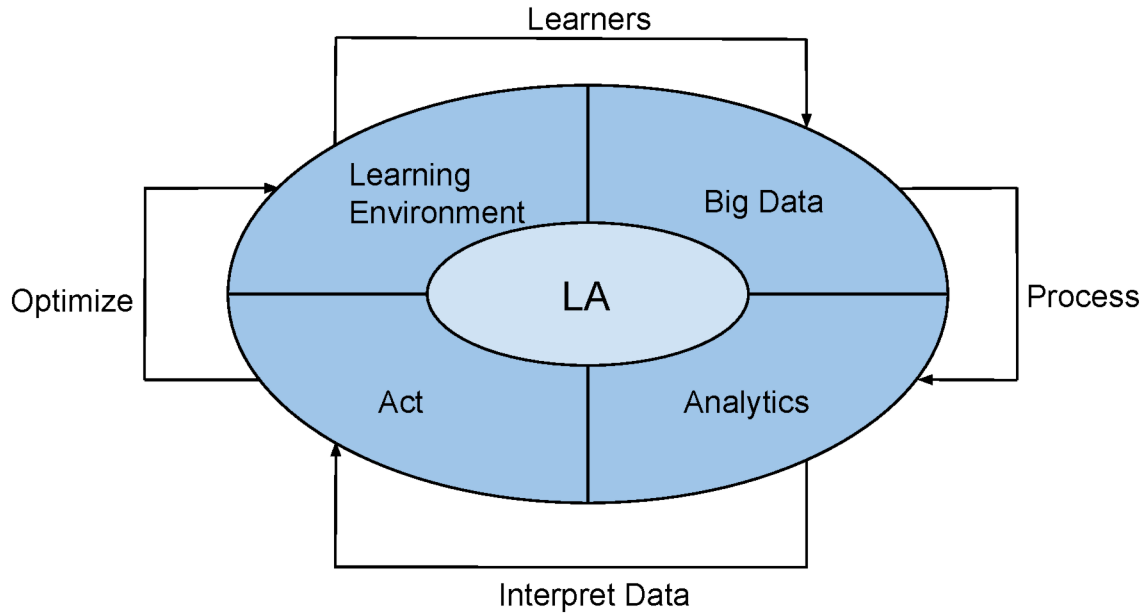


Figure 2.1: LA life cycle adapted from [KE15].

As an example, researchers like Heinemann et al. [HGD⁺24] developed a VR learning environment called RePiX VR which teaches the basics of computer graphics by explaining the Rendering Pipeline to the users [HGS23, HGS⁺22].

By collecting the generated learning data, they could investigate and evaluate the efficiency of interactive tasks.

These actions help to detect misconceptions as well as successful learning patterns leading to a better understanding of a learner's behavior [MG19].

The analysis of learning data involves an iterative cycle in which the data first gets generated from a learning environment that is then stored in a LRS, leading to an analysis of the retrieved data and taking action after the analysis outcome [KE15].

An illustration of this process can be seen in Figure 2.1.

LA is a systematic, interdisciplinary field that employs diverse methods and analytical techniques to assess the effectiveness of pedagogical approaches. By analyzing this data, LA helps enhance curriculum design and content delivery in higher education [JDT20].

In the context of LiMoAPI, the application was developed to monitor the incoming learning data in a LRS ensuring a better way for the verification of xAPI data.

2.2.2 LA in a XR Context

As already mentioned above, LA is an interdisciplinary field which also contains multiple diverse application fields.

In prior studies, it has been shown that the use of eXtended Reality (XR) brings various learning benefits, such as a deeper understanding of content, improved spatial concepts as well as improved long-term memory and increased motivation [WKS22].

This is due to the fact that users combine theoretical concepts with the interaction of their physical environment leading to a higher interaction with the virtual content [Rad14].

But there was no optimal way to include the use of LA in XR learning environments. This is a focus of the research group Learning Technologies which developed the OmiLAXR Ecosystem [HEGS22].

This Ecosystem contains multiple different components which facilitate the usage and integration of LA in VR contexts.

Due to the fact that LiMoxAPI is an extension of the OmiLAXR Ecosystem, the application was originally developed for VR-developers. Yet, also developers from other LA application fields have shown great interest in the concept of LiMoxAPI.

Therefore, this project strives to make LiMoxAPI accessible for LA developers in general without requiring an application field of XR.

2.3 Experience API

In order to achieve improvements in education, different tools are needed to retrieve the learning data. This can be done via the Experience API (xAPI) specification¹.

xAPI is a specification that describes how data is captured, stored and retrieved [VRL15]. It is organized in JSON data format and defines statements which represent the learning events of a Learning Management System (LMS) [BKP⁺16].

Each statement requires three different properties, being actor, verb and object and it composes sentences like "User1 (*actor*) clicked (*verb*) button (*object*)".

A concrete example statement is displayed in Figure 2.2.

The actor refers to the person who initiated the event being the learner, the verb captures the action or predicate of the statement, and the object can represent an activity, agent, group, or another statement. [VRL15].

A statement can be extended with a context, activity or result field for more extensive usage in which the context field provides additional information about the learning scenario specifics, the activity field contains detailed information about the target activity and the result field can be specified for a measured outcome [SLMOH⁺17].

This vocabulary can be defined and stored in central registries².

Registries serve as shared repositories, allowing other users to access previously created definitions [Väk23].

Definitions are stored in JSON format because this format strikes a balance between human readability and computer efficiency.

In order to store the produced statements, some kind of database is needed. xAPI uses a LRS which stores the statements in sequential order [SLMOH⁺17].

The xAPI standard³ defines an LRS as "a server (i.e. system capable of receiving and processing web requests) that is responsible for receiving, storing and providing access to Learning Records"⁴. A LRS is able to share the statements received with other LRSs and it can either exist on its own or inside a LMS [VRL15].

¹ <https://xapi.com/overview/>, accessed: February 2025

² <https://xapi.com/registry/>, accessed: February 2025

³ xapi-base-standard-documentation, accessed: February 2025

⁴ <https://xapi.com/learning-record-store/>, accessed: February 2025

```

{
  "actor": {
    "objectType": "Agent",
    "name": "John",
    "mbox": "mailto:testing@lti-lab.de"
  },
  "verb": {
    "id": "https://xapi.elearn.rwth-aachen.de/definitions/seriousGames/verbs/ended",
    "display": {
      "en-US": "ended",
      "de-DE": "beendete"
    }
  },
  "object": {
    "objectType": "Activity",
    "id": "https://xapi.elearn.rwth-aachen.de/definitions/seriousGames/activities/game",
    "definition": {
      "name": {
        "en-US": "game",
        "de-DE": "Spiel"
      },
      "description": {
        "en-US": "Represents a game or competition of any kind.",
        "de-DE": "Repräsentiert jede Art von Spiel."
      }
    }
  }
}

```

Figure 2.2: Screenshot of a possible xAPI statement consisting of an actor, verb and object

2.4 UI and UX Design

Although User Interface (UI) Design and User Experience (UX) Design are two separate concepts, they both play a crucial role in the development of digital products. UI design as well as UX design directly impacts how users engage with an application and affects their user experience [Ham23].

UI design makes up all visual components and interactive elements of an application. Developers have to figure out a specific layout, a color scheme, buttons and other visual elements that construct the user interface. It aims to produce an intuitive and user-friendly interface which is easy to navigate and interact with [RSS21].

However, UX design rather refers to the whole user experience while using the application. Therefore, developers try to create an enjoyable user flow while meeting the needs and expectations of the user [LRV⁺08, MHH18].

In order to fulfill these expectations, there are different key concepts to follow while developing a digital product:

For UI design, it is important to focus on the simplicity, consistency and visibility of an application while providing feedback to the user, e.g. a success-notification after a user has successfully logged in.

UX design, on the other hand, focuses on the user's goals and motivations for using a digital product and thus concentrates on the usability, accessibility, efficiency and clarity of an application [Ham23].

Following these principles and by focusing on stakeholder-centered design [For18], the UI of LiMoxAPI has been restructured and adjusted in order to achieve an overall better user experience and usability. This also sets the path for a more facilitated way of data validation.

Chapter 3 Related Work

This section presents the OmiLAXR Ecosystem which was developed by the research group Learning Technologies [HEGS22] as well as the Live Monitoring Tool LiMoxAPI [TSL24] which was developed to support VR developers with verifying the generation of learning data.

3.1 OmiLAXR Ecosystem

The research group Learning Technologies has developed an ecosystem that enables multiple stakeholders to work with VR learning environments [HEGS22]. It enables integrating the xAPI specification into educational VR applications [GHS24] and contains multiple components for different stakeholders:

OmiLAXR stands for “Open and modular integration of Learning Analytics in eXtended Reality” and aims to provide a simple integration of LA in XR environments for developers.

It also enables teachers and other educators to make use of LA by attempting to follow their student’s learning process whereas researchers strive to analyze the generated learning data that is stored in a LRS [Sie12]. Its components are presented in the following subsections.

3.1.1 xAPI Definitions Registry

The xAPI Definitions Registry stores VR specific definitions in JSON format. There was a seed vocabulary created by the research group which was based on a questionnaire, containing around 500 definitions [HEGS22].

The registry is versioned using a git repository. In order to enable more stakeholders to participate in the usage of the definitions, there also exists an xAPI Definitions Frontend which can be used by researchers and educators, but also for VR developers to search for specific definitions in the xAPI Definitions Registry or to suggest new ones.

A LA dashboard gives educators and investigators an insight into the learning process of their students and facilitates the decision-making process for learning activity and data analysis of stakeholders by visualizing the data [VODC⁺20].

Here, this xAPI Definitions Frontend is a first entry point when working with xAPI Definitions in which the definitions are also accessible via the corresponding URIs (Uniform Resource Identifier), making them easily accessible and recognizable.

The statements are presented in a more readable way for humans and therefore users are not required to have a computer science background in order to use the platform.

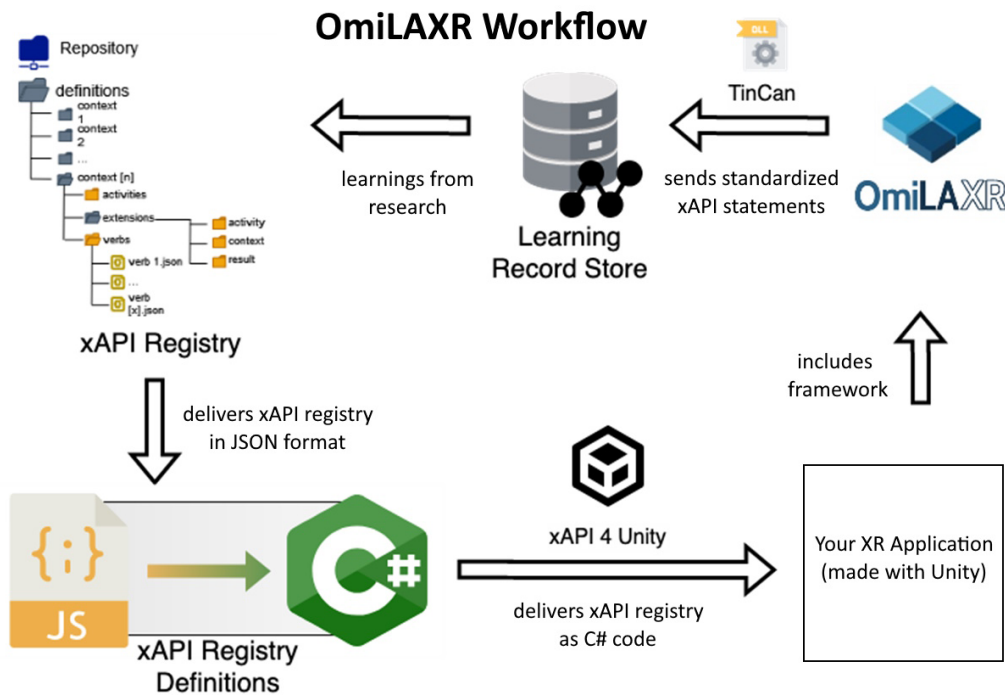


Figure 3.1: Workflow of the OmiLAXR Ecosystem, taken from [Gö24]

3.1.2 xAPI4Unity

Concrete xAPI statements have to be implemented for each specific learning scenario. This is done by VR developers which collaborate with different stakeholders. xAPI4Unity¹ serves as a support for developers by converting xAPI definitions from the xAPI Registry into C# classes [HEGS22]. This ensures a simpler synchronization of xAPI definitions.

3.1.3 The Researcher Companion Panel

The Researcher Companion Panel (ReCoPa) is a web tool which provides functionality to control the xAPI tracking settings and therefore enables stakeholders to work with the generated learning data. There is a constant live connection to a VR learning scenario, allowing users to select tracked variables for learning analytics or to store the information in a LRS [HEGS22].

3.1.4 OmiLAXR Framework

The OmiLAXR Framework² developed by [GHS24] was implemented to enable the integration of VR-specific statements into a VR environment, which have been defined with the development of the xAPI Definitions Registry [HEGS22].

The framework contains the XR Adapters System which supports the integration of third-party libraries by using the "Plug-and-Play" principle.

¹ <https://gitlab.com/learntech-rwth/omilaxr-ecosystem/xapi-4-unity>, accessed: February 2025

² <https://omilaxr.dev/>, accessed: February 2025

Apart from that, there is the Main Tracking System which facilitates different sub-tracking systems and generates the resulting xAPI statement.

A generated statement always contains information about what XR application was used, the game's version, if there is a specific game mode and the current task context. This statement is then sent to the connected LRS [GHS24].

3.2 LiMoxAPI

After looking at the OmiLAXR background, this subsection describes the idea and initial implementation of LiMoxAPI. The given details are referenced from [TSL24].

LiMoxAPI serves as an extension of the OmiLAXR ecosystem and has been developed to provide a lightweight, sophisticated testing tool for VR developers.

Usually when testing a learning environment, the connection to the LRS, here being LL, has to be established and has to be continuously reliable. However, this setup is complex and time consuming.

Due to the issues reported in the conducted study of the research group [GHS24], LiMoxAPI was developed to provide a simpler testing alternative.

LiMoxAPI stands for Live Monitoring of xAPI Statements and runs locally on the machine next to the learning environment it should monitor.

It periodically fetches the stored learning data from the connected LRS and visualizes it in its dashboard. An illustration of this is displayed in Figure 3.2³.

Being cross-platform makes LiMoxAPI suitable for multiple different users.

The subsequent sections explain the technical stack of the application as well as the core functionality of the four frontend components.

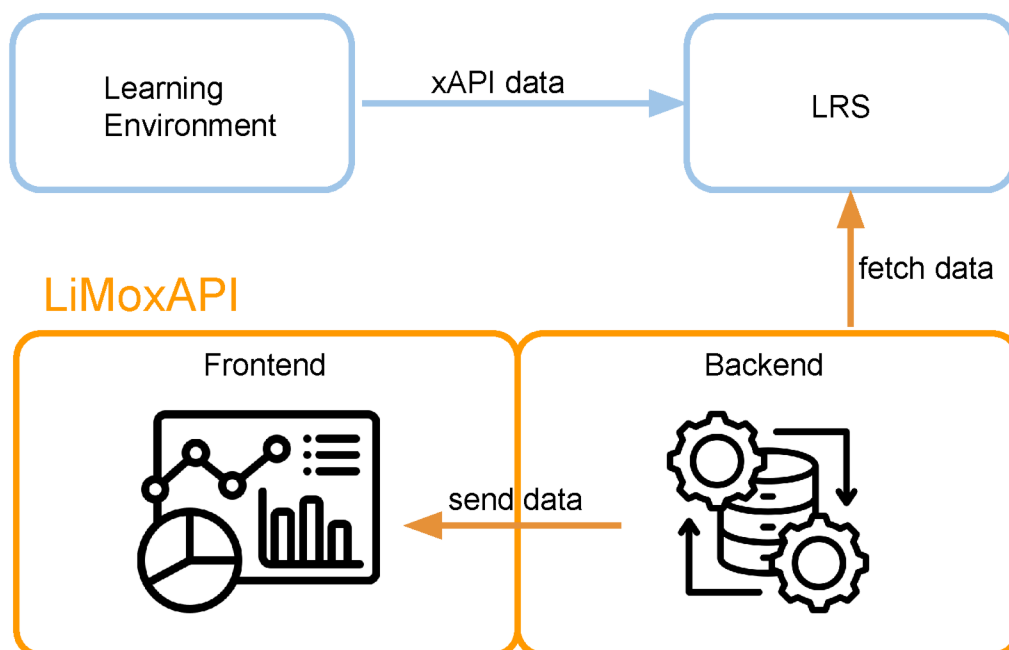


Figure 3.2: Concept Architecture of LiMoxAPI adapted from [TSL24]

³ This figure uses resources from <https://www.flaticon.com/>

3.2.1 Technical Stack

LiMoxAPI was developed using the Tauri Framework⁴ and can be divided into two logical parts: a backend and a frontend.

The backend is implemented in Rust⁵ which uses the Tauri ecosystem of libraries and periodically fetches the generated xAPI statements every 300 milliseconds from the connected LRS until the application is closed [TSL24].

Here, this LRS is the Yet Analytics SQL LRS⁶ which has to get started in parallel to LiMoxAPI. A pre configured version of this LRS is available in addition to the LiMoxAPI binary file.

The fetched statements are processed and formatted in the backend. The formatted data can then be used for the different components in the frontend which will be described in detail in the subsequent section.

In the frontend, the retrieved data gets visualized which is implemented in plain HTML, CSS and JavaScript with the additional usage of the Vue.js Framework⁷.

Each feature is implemented as a single component and handles the received data from the backend individually. The frontend listens to new incoming data from the backend and triggers according update functions ensuring synchronization between the backend and the frontend.

An illustration of the data flow in LiMoxAPI can also be seen in Figure 3.3⁸.

Both application layers are able to communicate via an event system channel: Tauri uses asynchronous message passing for the Inter-Process Communication (IPC) being *Events*⁹ and *Commands*¹⁰.

Events can be sent by both sides and are one-way messages which do not require responses. Commands, however, allow the frontend to call Rust functions in the backend.

This is done by sending a request which triggers an invoke handler to call the actual function. The request as well as the response is transformed into JSON format allowing to call functions of another programming language.

3.2.2 Frontend Components

The application's core functionalities make up four components being the *Statement Pulse Monitor*, the *Info Component*, the *Logging Console* and the *Filter Component*. The Login View and the Dashboard View of the application are shown in Figure 3.4 and Figure 3.5 respectively.

The Pulse Monitor can be seen on the top left of the Dashboard View and graphically displays how many statements are arriving at the connected LRS per second.

On its right side, the Info Component sums up basic information for the user. It shows the timestamp of the latest incoming statement, the total number of statements that have been monitored so far and at last it displays the amount of actors involved in the

⁴ <https://v2.tauri.app/start/>, accessed: February 2025

⁵ <https://doc.rust-lang.org/book/>, accessed: February 2025

⁶ <https://github.com/yetanalytics/lrsql>, accessed: February 2025

⁷ <https://vuejs.org/guide/introduction.html>, accessed: February 2025

⁸ This figure uses resources from <https://www.flaticon.com/>

⁹ <https://v1.tauri.app/v1/guides/features/events/>, accessed: February 2025

¹⁰ <https://v1.tauri.app/v1/guides/features/command/>, accessed: February 2025

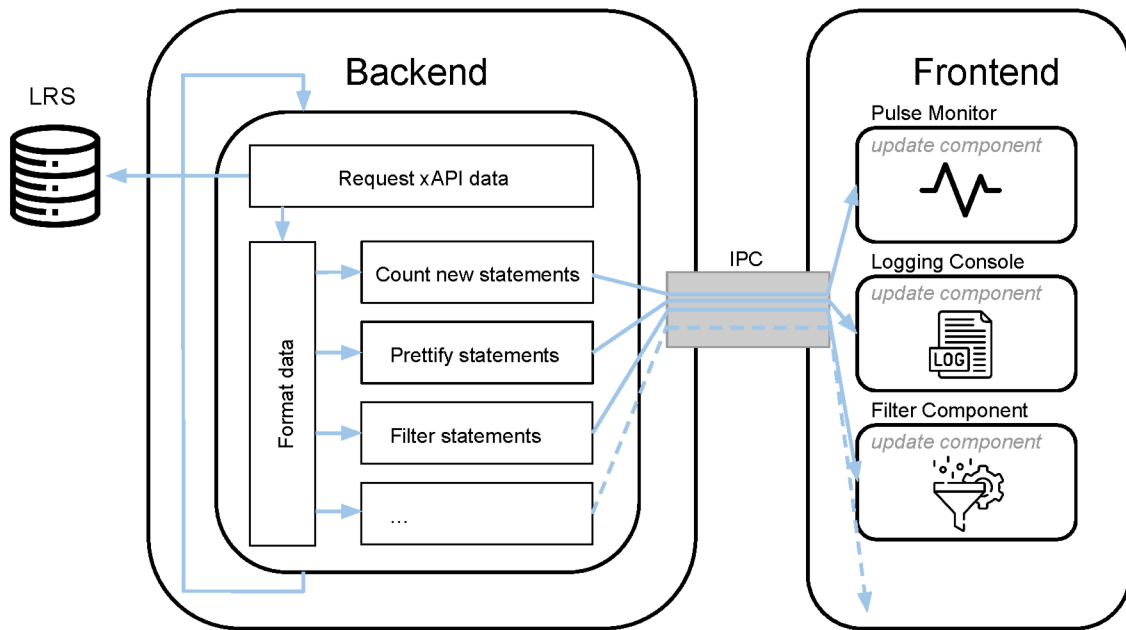


Figure 3.3: Internal Architecture of LiMoxAPI adapted from [TSL24]

current monitoring session.

Below the Info Component and the Pulse Monitor, there is the Logging Console.

In the Logging Console, the fetched statements from the LRS are logged by displaying the timestamp of the arrival, its actor name as well as the specified verb and object. By clicking on one of the logged statements, a dialog pops up and displays the full LRS entry in readable JSON format.

Lastly on the right side, there is also the Filter Component which allows the user to create own filters by specifying a path and a corresponding value. Whenever a statement arrives in which this value occurs under the specified path, a counter, which is displayed in the UI, gets incremented.

Providing a *path* and a *value* in a filter is defined as a *filter rule*. When multiple filter rules are specified in a filter, they are handled as ANDs. Therefore, the filter counter only increments if *filterRule1* and *filterRule2* both evaluate to true.

In addition, there is also the option to link a sound to a specified filter, such that the application plays a sound whenever a specified statement occurs.

This is due to the fact that usually there have to be two people involved in a VR environment testing process:

When VR developers try to test the generation of a specific statement, one usually has to wear a Head Mounted Display (HMD) whereas the other person has to manually check if this statement has been generated by refreshing the connected LRS.

This takes up time. With the filter component, a single developer is able to test their environment by awaiting an audible notification after the arrival of a statement.

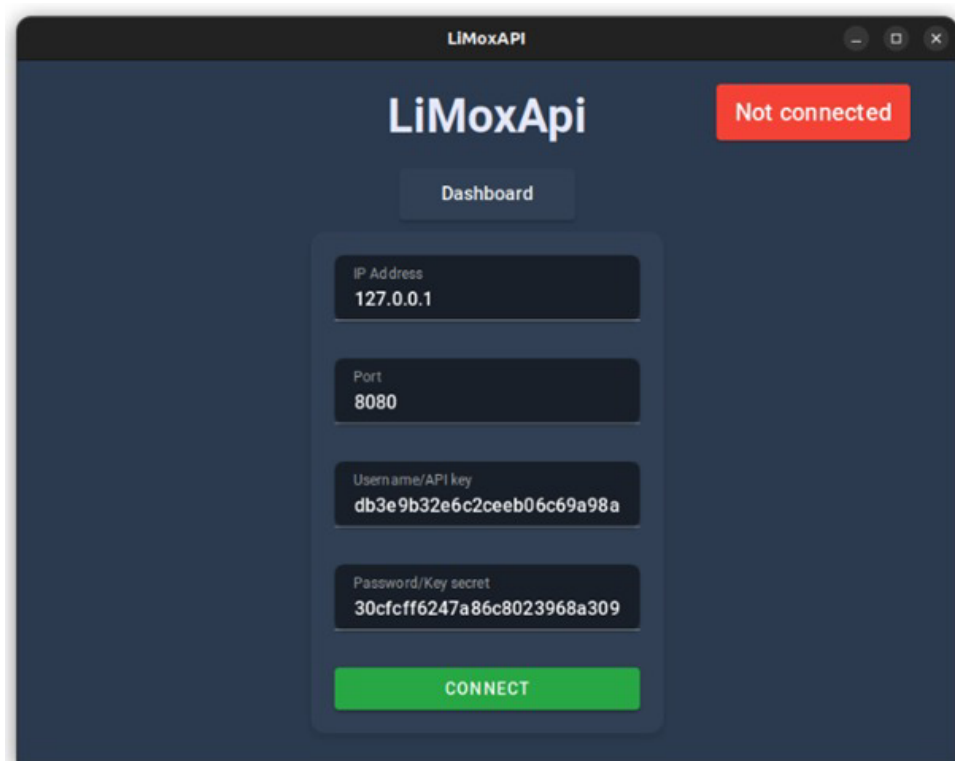


Figure 3.4: Login-View of LiMoxAPI taken from [TSL24]

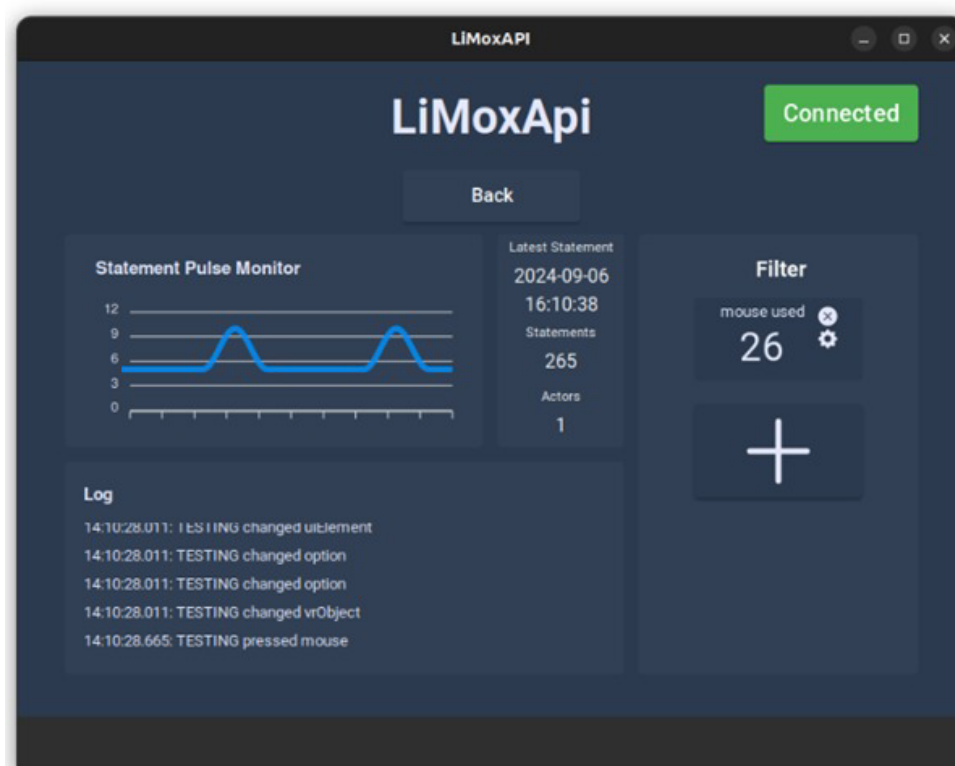


Figure 3.5: Dashboard-View of LiMoxAPI taken from [TSL24]

Chapter 4 First Requirements and User Evaluation

This thesis tries to answer the question, how the verification of the generation of learning data in LA environments can be even better ensured by developing graphical as well as logical improvements for LiMoxAPI.

Therefore, this chapter defines first requirements of the application and describes their implementation as well as illustrating a conducted user evaluation by regarding the setting, the study and its results. This evaluation was conducted to widen the perspective on the analysis of xAPI statements in order to retrieve additional requirements for the monitoring tool.

4.1 Reviving Requirements

In the prior bachelor thesis [TSL24], a final user evaluation was conducted in order to present the current state of the final application.

This evaluation has shown that LiMoxAPI provides a high-quality performance with the ability to process up to ten incoming statements per second which is more than sufficient when monitoring learning environments.

The participants of the study were overall content with the outcome of the application, still some additional requirements were defined.

One mention was that LiMoxAPI needed a better interplay between its different components. One way to ensure this is by connecting the filter component and the logging console:

The user should be able to only log filtered statements in the logging console and switch between the two different logging states via a checkbox which is displayed in the log.

On the other hand, the user should also be able to directly select filters from the logging console which should be applied as filters.

This is realized via checkboxes which are located next to each logged statement. When a user selects a statement, a function is called in the backend which retrieves the corresponding values to the actor, verb and object path of the statement. Next, a new filter is created which includes these three filter rules.

Another requirement that was pointed out was that the user had no possibility to restart the current monitoring session. This led to the development of a reset button which resets all filter counters to zero, empties the log, deletes the current graph in the pulse monitor and sets the given information in the info component to zero.

In addition, a pop up dialog was added which asks the user to specify which filters he

wants to keep.

This way the user can decide if he either wants to keep the filters that he manually created via the filter component or if all filters should be kept, including the selected filters from the logging console.

An aspect which was not reported in the prior project was that the scroll functionality in the logging console was buggy.

This was adjusted such that the logging console automatically scrolls downwards when new statements arrive until the user scrolls up. When this happens, the scrolling stops and only restarts when the user scrolls down again.

Other improvements for the user experience were achieved by adjusting LiMoxAPI with a more coherent structure: All buttons have the same positioning, color and hover effect and the overall application now has a responsive styling.

Lastly, some useful error notifications have been added which inform the user about incorrect actions guaranteeing an improved data integrity.

The listed filter items are shaking and displaying an error message, whenever the user selects a statement as a filter which already exists in the filter list.

The same behavior appears when a user tries to only log filtered statements, although no filters have been specified.

Thus, the user knows when he is taking incorrect actions and therefore gets a clearer understanding of the whole application.

4.2 Evaluation

After having implemented the first requirements as defined above, a user evaluation was conducted in order to discuss the current state of LiMoxAPI and to define additional needed features.

For this sort of project, it is rather difficult to find participants for user studies because they are time-consuming and potential participants have to actively work in a field with a clear reference to LA.

G. Thiesen [TSL24], who has initially worked on LiMoxAPI, conducted an evaluation with two VR developers who were both using the OmiLAXR Ecosystem.

For this study, two developers from different LA application fields could be found to participate which is very fortunate because neither of them uses the OmiLAXR Ecosystem.

The aim for the evaluation was to get an insight into LA outside of VR development and gain another perspective on LiMoxAPI, which could luckily be achieved.

4.2.1 The Setting

The user evaluation took place in an online meeting and was conducted in two separate sessions such that each participant was interviewed alone.

By dividing the participants, it was ensured to get to know their research fields more intensely and to get their unbiased opinions and suggestions for the application.

Before the evaluation started, they had installed a binary of LiMoxAPI on their computer and had connected it to the provided pre configured YetAnalytics SQL LRS.

The evaluation lasted around 90 minutes and followed the cooperative evaluation approach [Hol93].

It started with a short motivation of the application, followed by five different exercises the participants had to solve by using LiMoxAPI. Each exercise confronted them with one component or the interaction between multiple components.

After the five exercises, possible additional features were portrayed and discussed. The following subsections will contain a description of the tasks as well as the results of the study.

4.2.2 The Study

The participants were given the following tasks:

1. Look at the different components. What do you think each of them does?
2. Create a filter which counts how often the user uses a mouse and only log these filtered statements to the logging console.
3. Display one of the logged statements in full JSON format.
4. Restart the monitoring session, select a statement as a filter from the logging console and count how often this statement does *not* occur. Whenever this happens, play Sound 3.
5. Experiment with some filters from the logging console and then try to reset the monitoring session. How would you evaluate the reset button functionality?

Subsequent to each task, a small discussion was held including the following questions:

- What was unintuitive about the task?
- What was difficult or unclear?
- How could this specific feature be improved for a better user experience?
- Which functionality do you miss in this component/in LiMoxAPI in general?

After these five exercises, the participants have gotten to know the application quite well and had used every existing feature so far.

Subsequent to the tasks, new considerations for potential features were presented and discussed with each participant.

4.2.3 The Results

During the study, the participants mostly showed the same behavior and reacted equally to the given tasks. Therefore they also reached the same problems and came up with similar enhancement ideas.

Both participants liked the layout of the whole dashboard as well as the idea of each component. It was easy for them to understand the basic functionality of the logging console, and they had no problem adjusting the settings of a given filter.

In addition, they agreed that the feature of only logging filtered statements was very important as well as the reset functionality of a monitoring session.

Still, some issues could be retrieved which will be presented in the next section.

Defined Problems

For both participants it was unclear in which time unit the pulse monitor displayed the graph of incoming statements and what exact numbers the facts in the info component referred to.

Additionally, they stressed that the pulse monitor showed too few data points and should provide an option to display a graph within the range of multiple minutes.

When looking at the filter component, it was mentioned that "Filter" was an unfitting and confusing name. Moreover, the creation of a filter was unintuitive and it was difficult to discover how to properly define filter rules.

Although both participants stated to like the basic functionality of the filter, both of them wished to have more complex features.

In the logging console, the checkboxes to select statements as a filter were mistaken as bullet points. Apart from that, the display of the full xAPI statement in JSON format was not found on their own. They had to be actively guided to click on a logged statement in order to show the full LRS entry.

At last, the participants had problems to understand the dialog in the reset button which asks to keep the given filters. Both of them commented that the formulation and definition of different filters was unclear and the user would not understand what he was asked for.

The subsequent sections contain the outcome of the discussion at the end of the study as well as the final retrieved requirements.

Final Discussion

Subsequent to the tasks, new considerations for potential features were presented and discussed with each participant.

It turned out that the filter component was quite useful and important, therefore it was a high priority to increase the complexity of the filter functionality:

Both participants stressed that the enablement of RegEx in the value field of a filter rule was highly demanded.

Apart from that, the filter should allow a higher complexity between different filter rules by allowing complex filter logic between them. Possible operations should be AND, OR, XOR and NOT.

Apart from filter extensions, the idea of a download button was presented. This download button should download the given login credentials from the used LRS such that users could store the credentials for multiple LRS's locally on their computer. Both participants agreed that this feature would be useful.

The Idea of Creating a Plugin System

Another idea of this thesis, next to the focus on UX enhancement, was to provide a plugin system in LiMoxAPI such that users would be able to implement their own features for their own specific use case.

This could have made LiMoxAPI accessible to a wider range of users due to the ability of customizing the application.

Instead, the user evaluation has shown that a plugin system would rather decrease the willingness of using LiMoxAPI. In order to be able to use the plugin system, users

would have to learn the Rust programming language which would have been a great burden. Rust is quite difficult in the beginning of the learning process, thus it would be too much of an effort to make use of the plugin system.

The participants argued that this feature would rather be used for research purposes when conducting user studies for their learning environments instead of the development process of their applications.

Therefore, this thesis rather focused on enhancing the already existing features in the UI and on enhancing the overall user experience.

Retrieved Requirements

After seeing the problems and discussions that arose during the user evaluation, this section will present the requirements which could be retrieved from the outcome of the study.

In order to provide a better understanding of the pulse monitor component, an x-axis labeling has to be added. Currently, the pulse monitor requests the number of new incoming statements from the backend every 300 milliseconds.

Instead, it should request the new amount every 250 milliseconds but only label every fourth data point which corresponds to every second. This provides a clear time unit to the user and makes the whole component more comprehensible.

In order to enable the user to look at older values of the displayed graph, a history log should be implemented which holds the number of incoming statements from the last fifteen minutes in a table.

The displayed facts of the info component have also been confusing without a given context. Here, an additional explanatory text should be displayed when clicking on an according icon.

Regarding the filter, the component should be renamed into "Statements Counter" because the filter is completely independent from the other components and rather counts the occurrence of specific statements than interfering with the other features. Apart from that, an explanatory text as well as a dropdown menu in the path field of a filter rule should be provided in order to guide the user through the creation of a filter.

In addition, the filter needs more complex features like a search for RegEx values and an option to provide a filter logic for the given filter rules. In order to be able to analyze single filters, a reset button should be provided in each filter item in order to reset the statements counter of a specific filter.

Moving on to the logging console, a whole restructuring is needed.

The log has to be displayed as a table instead of a list. When the checkboxes are displayed in a separate column, they will be recognized as checkboxes and not as bullet points.

In order to show the user the option to click on a logged statement, statements should get highlighted and the cursor of the mouse should change to a hand-icon when hovering over a logged item which implies that the statement is clickable.

Apart from that, the reset button needs a clearer formulation and more consistent definitions of the different filters.

And lastly, a download button has to be implemented in the Login View of LiMoxAPI with which a user can download the entered credentials of the connected LRS.

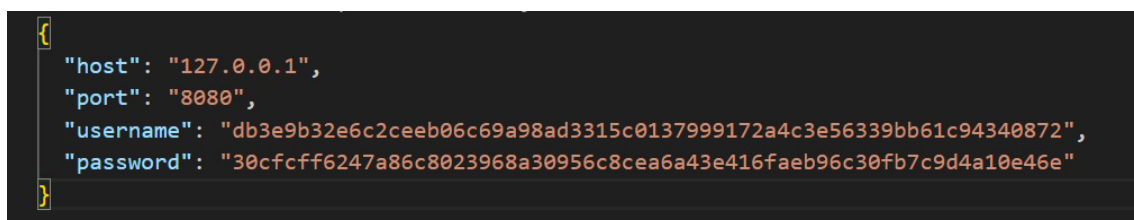
Chapter 5 Implementation

As seen in the previous chapter, LiMoxAPI is in need of a better UI design and brings a high potential to provide more complex features which enhance the interactivity between the components and the user. These changes also imply that users will have better options for verifying and monitoring their statements. In addition, the enhanced user feedback and an overall improved clarity of the application ensures a better data integrity for the user.

This section describes the concrete implementation of the different features which followed from the evaluation.

5.1 Download and Reset Button

In order to allow the user an easier way of managing their login credentials for different LRSs, a download button was added to the login view of the application. It is located right next to the login field. By clicking on it, the browser of the computer opens and the user can specify a path where the *credentials.json* file should be stored. The *credentials.json* file looks like the following and contains the current host, port, username and password of the used LRS:



```
{  
  "host": "127.0.0.1",  
  "port": "8080",  
  "username": "db3e9b32e6c2ceeb06c69a98ad3315c0137999172a4c3e56339bb61c94340872",  
  "password": "30cfcff6247a86c8023968a30956c8cea6a43e416faeb96c30fb7c9d4a10e46e"  
}
```

Figure 5.1: Example of the downloaded Login Credentials stored in JSON format

In addition, the reset pop up dialog was adjusted by clearly defining the different kinds of filters that can be created in LiMoxAPI:

There are the *selected filters* which are filters that have been selected via a checkbox in the logging console, and there are the *custom filters* which refer to filters that a user has created manually in the statements counter component.

The pop up dialog in the reset button now gives the options to either keep all selected filters, keep all custom filters or keep all existing filters as well as keeping none of them. The resulting dialog is displayed below.

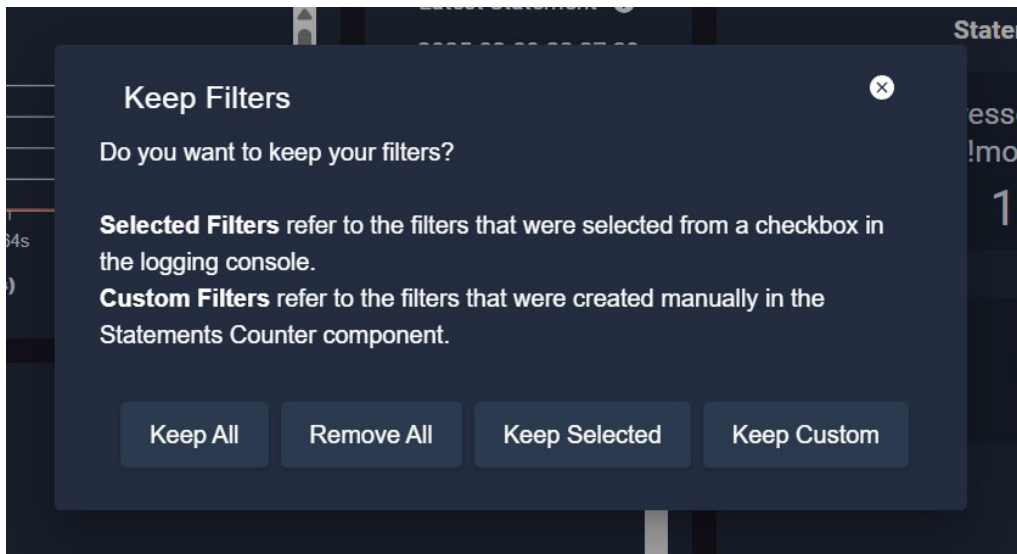


Figure 5.2: The new Reset Dialog Formulation

5.2 Helpful Explanations for an improved UX

The study has shown that it is tremendously important to provide helpful explanations and feedback to the user in order to achieve an enjoyable usage of the application. Therefore, different explanatory texts have been added to almost every component which can be accessed via clicking on a question mark icon.

One of these icons was added into the filter settings dialog being located next to a filter rule. It contains instructions on how to create a filter rule by showing in which way the different paths should be provided.

Additionally, a question mark icon was added to the info component which provides clear definitions for the displayed information.

Apart from that, the pulse monitor also contains additional explanations on how often the statements are fetched from the LRS and on the range as well as the interpretation of the displayed graph.

5.3 Pulse Monitor

The question mark icon was not the only thing which was added to the pulse monitor. Two adjustments have been made, being the x-axis labeling of the chart as well as adding a history log of the past 15 minutes. These two features will be described in the following subsections.

5.3.1 X-Axis Labeling

It has been criticized multiple times that the whole component was quite ambiguous and the chart with the graph definitely needed an x-axis labeling to ensure that users would be able to interpret it correctly.

The chart displays a graph which is updated every 250 milliseconds, but only every fourth data point is being labeled such that the x-axis is displayed in seconds. The overall range of the portrayed graph amounts to 10 seconds.

Whenever the monitoring session is reset, the x-axis restarts at zero.

5.3.2 History Log

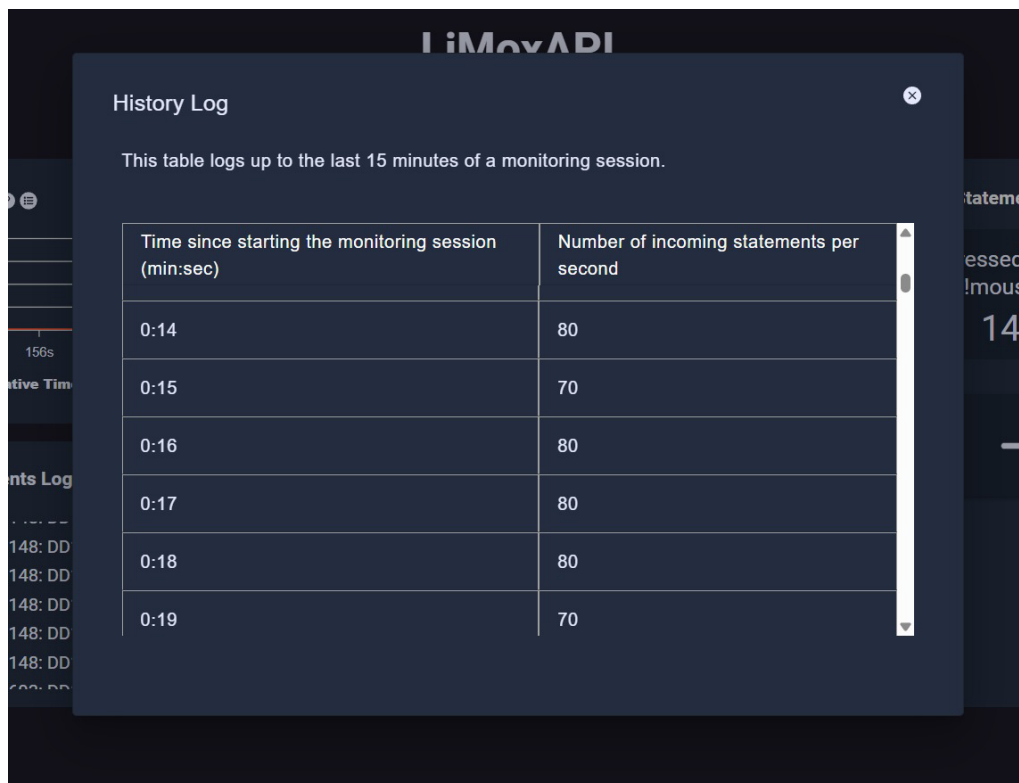
The history log can be accessed via the log-icon next to the help-icon which displays further explanations. It is displayed as a table with two columns.

On the left, there is the amount of time in the format min:sec and the right column displays the amount of statements that arrived at that time.

After 3600 entries which correspond to 15 minutes ($15 \cdot 60 \cdot 4 = 3600$), all prior entries get removed from the table. This is done due to performance measures.

In general, the ApexCharts library is not fully suitable for this kind of chart. When dealing with large datasets, especially in real-time scenarios like LiMoxAPI, ApexCharts often struggles with performance, which can cause laggy interactions and slow rendering times.

Displaying new data every 250 milliseconds, causes a high overhead which is also why the x-axis only labels every fourth data point.



Time since starting the monitoring session (min:sec)	Number of incoming statements per second
0:14	80
0:15	70
0:16	80
0:17	80
0:18	80
0:19	70

Figure 5.3: The History Log of the Pulse Monitor

5.3.3 Additional Statement Type Graphs

One participant of the evaluation also demanded a feature which would enable the user to display the amount of different statement types in the pulse monitor.

This participant defines different statement types via the fields *object.definition.type* and *verb.id* which can be provided in xAPI-statements.

Nevertheless, the implementation of this feature has shown to be quite tedious because the field *object.definition.type* has no uniform structure in his statements. Depending on each object type, this field provides different constructions.

Apart from that, the performance issues in the ApexCharts chart make it almost impossible to enable the user a sophisticated second graph-view for his analysis. This is why the implementation of this feature was aborted.

5.4 Logging Console

Looking at the logging console, the highest priority was to portray the given features more obvious to the user. That included the checkboxes next to each logged item with which one can select a statement as a filter as well as the ability to click on a statement to display it in full JSON format.

This was achieved by restructuring the logging console from a list to a table with two columns. The column on the left now says “Select as Filter” containing all checkboxes whereas the column on the right has the header “Statements Log” containing the logged statements as its items.

In addition, the cursor of the mouse now changes to a hand-icon when hovering over the logged statements. The statement which is hovered over, also gets highlighted, telling the user that this item is clickable.

When hovering over a checkbox, the tooltip “Select Statement as Filter” gets displayed. There has also been the discussion if tooltips should be added to the logged statements, displaying a text that implies that the user can display the full statement with a click.

In the end, the idea got canceled because the tooltips could have been disturbing to the user depending on the amount of statements being logged, especially when scrolling through the log in general.

The overall resulting dashboard looks like this:

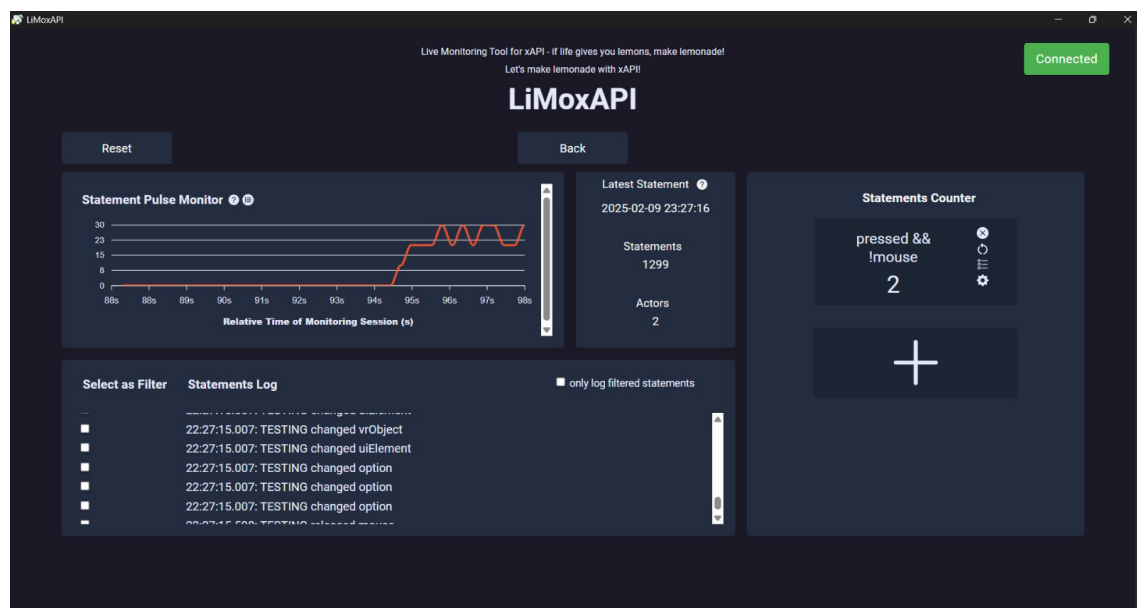


Figure 5.4: The final Dashboard-View of LiMoxAPI

5.5 Filter Enhancements

The improvements in the different components already had a great impact on the overall usability of LiMoxAPI, but the highest focus by far lay on the filter component. Multiple suggestions could be extracted from the evaluations and both participants showed great interest in this feature.

5.5.1 Simplified Filter Rule Definition

The evaluation has shown that the creation of a filter, especially the definition of a filter rule, has been extremely complicated. It was vastly unclear what was expected in the path field in a filter rule.

The added help-icon already gives instructions on how to create different rules but in addition to that, a dropdown menu has been added to the path field displaying example paths to filter. By clicking on the path field, frequent example paths are presented and the user can either choose between the provided paths or enter their own.

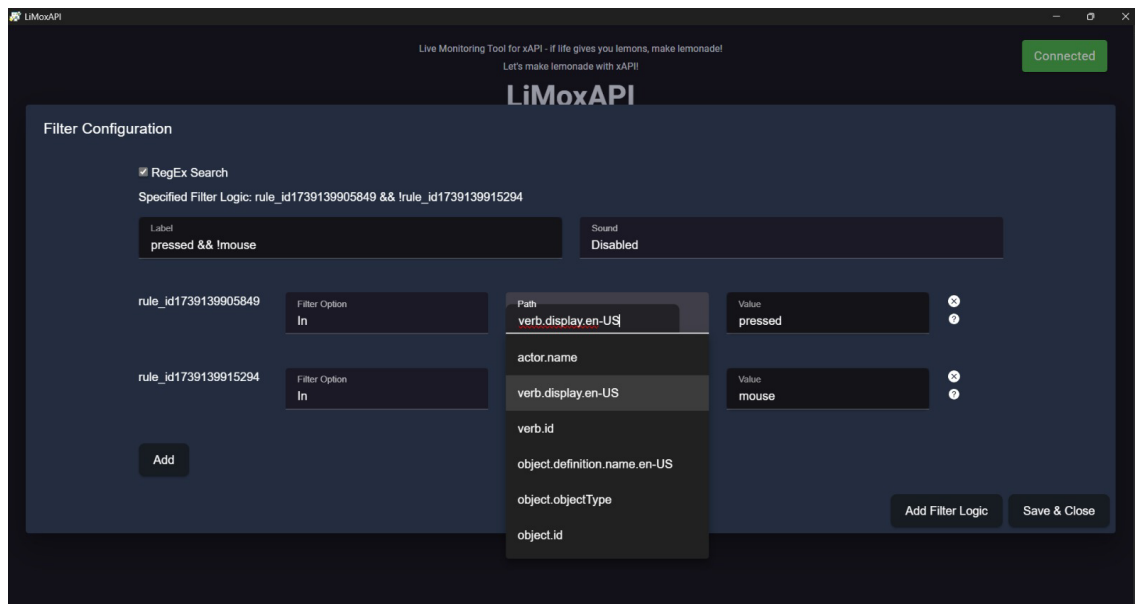


Figure 5.5: The Filter Settings Dialog with the added Dropdown Menu, RegEx Search and Filter Logic

5.5.2 Analysis of Single Filters

Due to the simplified filter rule definition and the provided instructions about the filter functionality, the user already gains a better comprehension about the overall component.

Apart from that, the user should also be able to look at different filters more closely and therefore have the ability to analyze the specified filters more individually.

In order to achieve this, a log-icon was added to each filter containing a list of all the statements which have been filtered by this filter.

Although the logging console already provides the feature to only log filtered statements, users now get an insight into the filtered statements separately and thus can

tell which statement counters get incremented by which statements.

For the purpose of having more control over each filter, users now also have the ability to reset the statements counter of a specific filter via a reset button in each filter item.

These two features enhance the analysis ability of a user and therefore supply a better way of monitoring the generation of learning data.

An example of the filtered statements log for the filter *pressed && !mouse* is displayed below.

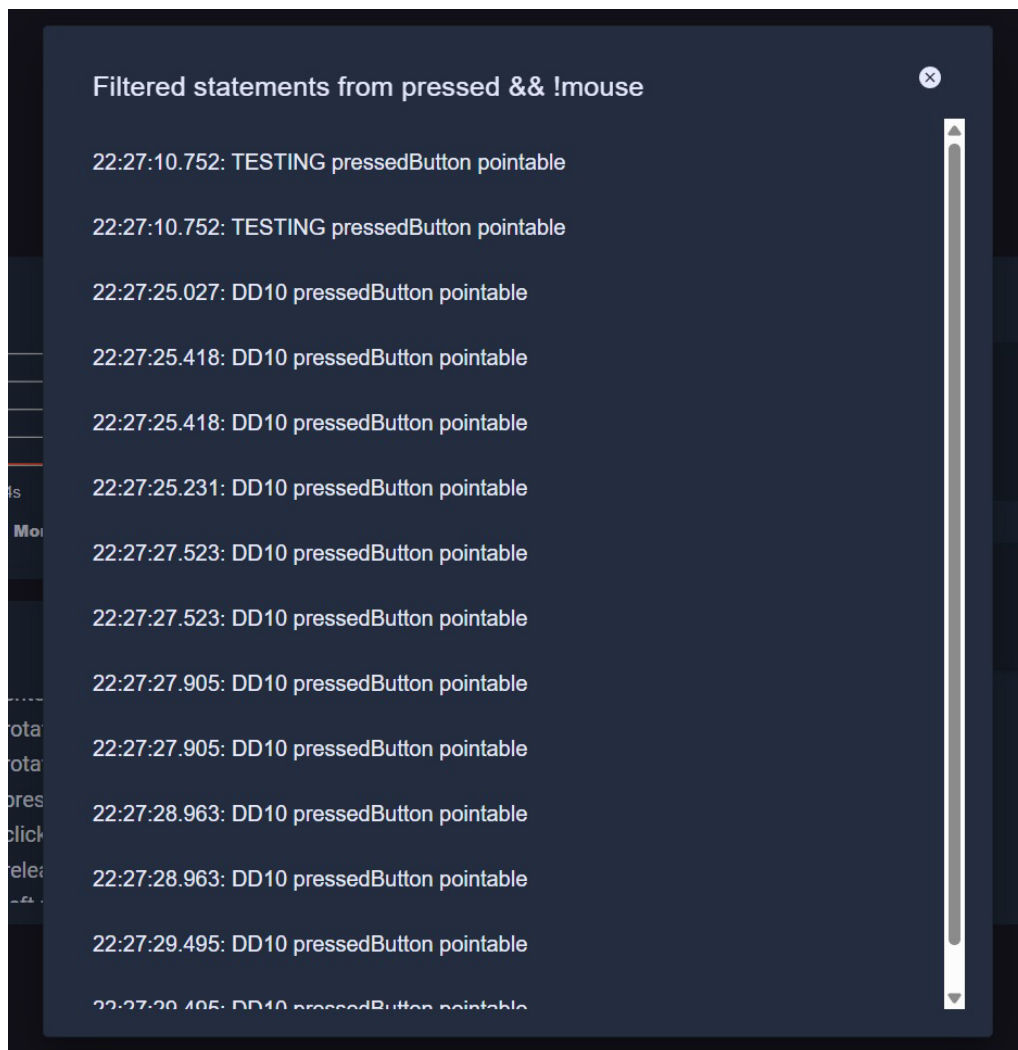


Figure 5.6: The Filtered Statements Log of Single Filters

5.5.3 RegEx

When creating a filter rule, defining the value field can be rather tedious. If a user wants to filter a specific verb-id, the filter rule could look like this:

Path: *verb.id*

Value: <https://xapi.elearn.rwth-aachen.de/definitions/seriousGames/verbs/ended>

As one can see, the provided ids can get quite long and it is exhausting to always copy them into the filter settings dialog.

Thus, the dialog now includes a checkbox which says "RegEx Search" providing a way to enter RegEx expressions in the Value fields.

RegEx is short for *Regular Expression* and it is a pattern which describes a set of strings that matches the pattern. In other words, a RegEx accepts a certain set of strings and always rejects the rest [Fri06].

So in the example above, the user could also enter *ended* into the Value field and it would match the correct pattern.

There are also more complex options to use this feature but the given example portrays the most frequent use case.

5.5.4 Filter Logic

The highest demand in the filter component was the ability to combine different filter rules with complex logic. By default, filter rules are always connected via AND operations and the aim was to also enable the operations OR, NOT and XOR.

In the evaluation, there have been broad discussions on how this could be realized. One idea was to add a dropdown menu between each filter rule such that the user can select an AND, OR or XOR operation between the rules. But then it would have been difficult to realize filters with more than two filter rules, because the user had no possibility to place parentheses in between. Moreover, it would not be possible to realize the NOT operation.

Instead, every filter rule now receives a unique *rule_id* which is displayed to the left of each filter rule. The user can then use the corresponding ids to specify a filter logic by clicking the button "Add Filter Logic" which opens a dialog with an input field.

An example could look like this:

If a user wants to filter how often the actor has either *pressed* (rule_id123) the *mouse* (rule_id456) or *clicked* (rule_id789) the *mouse* (rule_id456), the corresponding filter logic would be (rule_id123 && rule_id456) || (rule_id789 && rule_id456).

The resulting dialog to define filter logic looks like this:

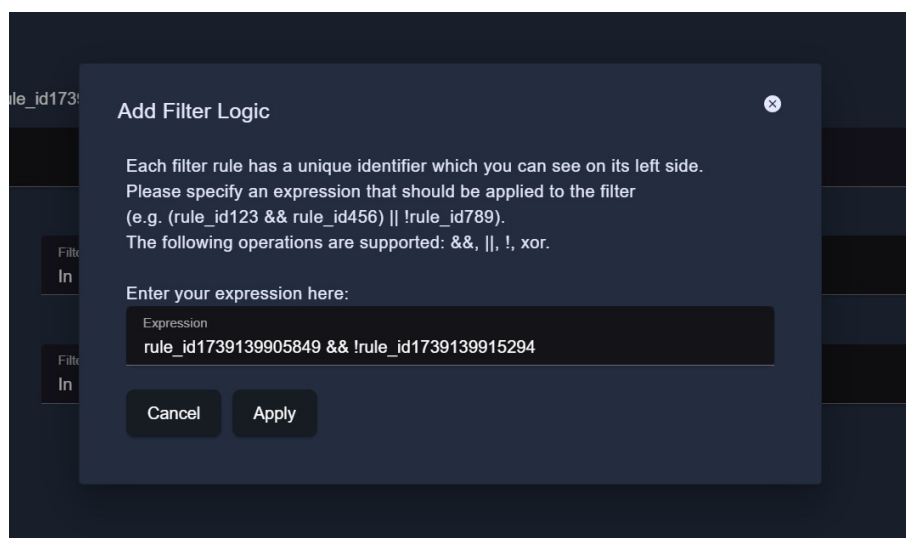


Figure 5.7: Add Filter Logic Dialog

Frontend Checks

The specified expression is displayed in the filter settings dialog, thus the user is always aware of their provided logic.

When a filter expression is entered in the input field and the user presses the “Apply” button, the frontend performs an initial check and makes sure that the provided expression is valid. It checks whether the provided rule_ids exist as well as checking if only valid characters were used, being “&”, “|”, “xor” and “!”.

At last, it checks if the amount of used parentheses is balanced, thus if the amount of opening parentheses equals the amount of closing parentheses.

If any of these checks fails, an according error gets printed to the UI, otherwise the filter logic gets applied and the backend gets called with the provided expression and the corresponding filter_id.

Backend Handling of Filter Logic

In the backend, there is a global *FILTER_EXPRESSIONS_MAP* which stores filter_ids with their corresponding filter expression as a string.

When the backend is called with a filter_id and a logical expression, it checks if the filter_id already exists in the map and either updates the expression, inserts a new entry or deletes it.

The function which is responsible for updating the filter counters contains an if condition which checks if a filter logic exists for the given filter. Depending on the outcome, either the usual function *filter_statements* or the new function *filter_statements_with_logic* gets called.

The new function starts with parsing the given expression string such that each operator and rule_id gets separated and stored in an array.

The string “(rule_id123 && rule_id456)” would now become [“(”, “rule_id123”, “&&”, “rule_id456”, “)”].

After that, for each filter rule it is computed, whether the provided path contains the specified filter value in each statement.

The boolean result for each statement is stored in the *rule_result* array. After filtering all statements with a filter rule, the rule_id as well as the rule_result array are stored in the *rule_results* map.

At the end, the map contains every existing rule_id with the corresponding boolean results from each statement. Next, these results get applied on the parsed expression:

Therefore, the helper function *evaluate_expression* is called for every index of the incoming statements. In this function, two different stacks are created being the *operator_stack* and the *bool_stack*.

The function loops through every parsed token and sorts the operator symbols, being ||, &&, xor and !, into the operator_stack. Whenever it identifies a rule_id, it looks up the rule result for the current statements index and pushes it onto the bool_stack.

When reaching an opening parenthesis, nothing happens. But when reaching a closing parenthesis, the helper function *apply_operator* is called with the two current stacks.

At that time, the operator_stack contains all operators which occurred in one pair of parentheses, thus they can be evaluated in one step. The bool_stack on the other

hand, contains the results of the corresponding `rule_ids` from that statement. The helper function now pops every item from the `operator_stack` and performs the corresponding operation with the according popped items from the `bool_stack`. The resulting boolean gets pushed onto the `bool_stack`, evaluating the expression backwards.

If there are any tokens outside of a parenthesis, `apply_operator` is called as well.

This process is repeated until the whole expression has been evaluated. The overall end result is stored in the `bool_stack`.

If the result is `true`, the corresponding statement is pushed into the `valid_statements` array, which is returned at the end of the function `filter_statements_with_logic`. These statements are then counted and the statements counter for this filter is getting updated.

This whole process is repeated for every statement that is arriving from the LRS as long as an expression is provided in the filter.

A detailed example is displayed below, given an expression, two filter rules and one fetched statement (Figure 5.8).

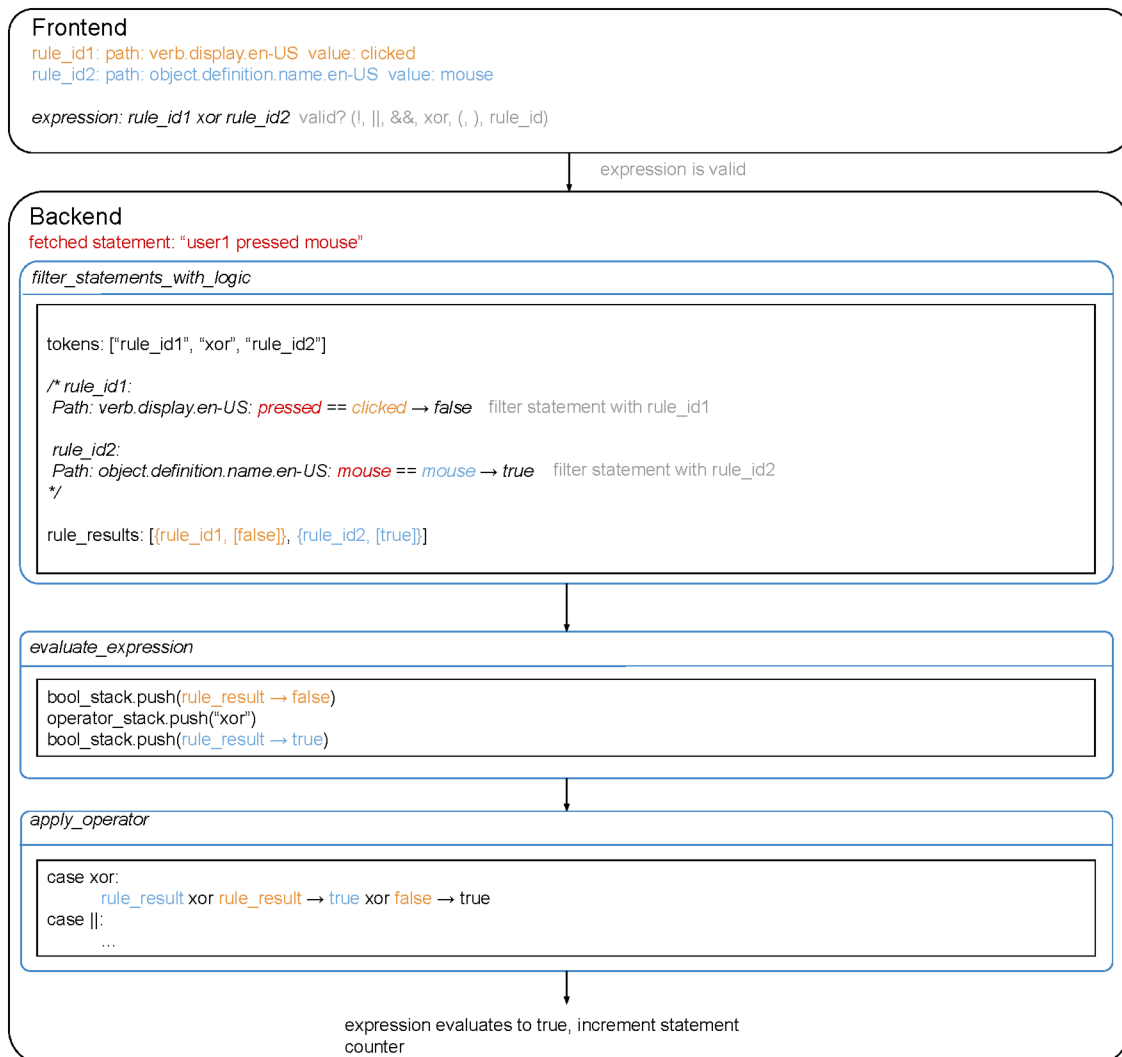


Figure 5.8: Simple Filter Logic Example

Chapter 6 Final Evaluation

At the beginning of this project, two researchers have been invited to take part in a user evaluation of LiMoxAPI. In this evaluation, the participants got to know the application quite well due to actively solving different tasks in the application.

As a result, the researchers gave their opinions about LiMoxAPI and suggested additional features and further improvements.

After the implementation of their suggestions, a second evaluation was conducted with the same participants. This time, the study was held as a group session instead of single meetings and the participants did not have to engage with LiMoxAPI themselves but rather followed a presentation of the final application and its new features. The results of the evaluation is presented in the subsequent sections:

6.1 The Evaluation

Just as in the first user evaluation, each component was portrayed and discussed individually without looking at the whole application at first.

For every component, the results from the first evaluation were presented followed by a description of how the suggestions got implemented, leading to a live presentation of the improved component in LiMoxAPI.

That way, the participants also learned about the concerns and comments of the other person during the first evaluation and could then immediately see the implementation of the suggested features and improvements.

During the evaluation, the participants could offer suggestions and ask questions at any time.

6.1.1 Pulse Monitor

The study started with the pulse monitor. The added features were the labels on the x-axis for every second, as well as the history log which portrays the amount of statements that arrived each second for the last 15 minutes.

The one thing that was not implemented after the evaluation was the second monitor view, which graphically portrays the number of different statement types arriving per second. This is due to the fact that the definition of different statement types does not follow a clear structure and the view would only be adaptable for the statements of this specific participant. Other users probably do not organize their statements in this way, therefore this feature would not find a broad application.

Both participants found the x-axis labeling really useful and necessary for a sophisticated analysis. The same goes for the history log. Although both of them said it would

be even nicer to expand the history for a greater amount of time, this is quite hard to realize with the current use of the ApexCharts library. When expanding the history log, one would forfeit the current performance of the pulse monitor and thus forfeit the performance of the overall application.

Another comment on the history log was that it would be nice to portray the log as a histogram instead of a table. This could give the user a simplified overview of the overall monitoring process.

6.1.2 Info Component

After regarding the pulse monitor, the evaluation focused on the info component. The only change in the info component was the added help-icon which displays additional information about the provided data. Both participants complimented the additional explanations and agreed that these help-icons highly contribute to an improved user experience and increased the comprehension of the user while using LiMoxAPI.

In general, multiple tooltips have been added to the icons in the application, which made a big difference in the UX as well. Both participants were vastly fond of that.

6.1.3 Filter Component

Subsequent to the info component, the evaluation went on to the filter. This component had by far the biggest changes, including the additional reset button for each statement counter, the simplified way to create filter rules due to the provided drop-down field with example paths, the enablement of RegEx search, the option to provide complex filter logic as well as the log of statements which were filtered by that specific filter.

The participants quite liked the new features and especially found the log of the filtered statements useful.

This feature did not evolve from the first user evaluation but rather became an idea during the implementation process while testing the new improvements. There was no option for users to specifically look at statements from a particular filter which can be particularly useful when complex logic is provided.

This is how the idea for this feature was created and both participants vastly agreed on this opinion.

Again, they stressed the importance of the provided explanations via the help icon of the component and were quite content with the outcome of the filter functionality.

Nevertheless, one thing both of them noted was the fact that the rule_ids which are needed for the provision of logical expressions are too long and would be exhausting to use. Right now, the ids get uniquely created and represent the timestamps of their creation.

Their suggestion was to use a pseudonym generator, because random names would be easier to remember than using a combination of multiple digits.

Moreover, one participant commented that the implementation of a saving and importing feature for the filters would be useful as well. With this possibility, users would be able to reload their old filters from prior monitoring sessions and could reuse them and their eventually provided logic saving a lot of time.

6.1.4 Logging Console

Next, the evaluation looked at the logging console. There, the only massive change was the display of the log. Prior, it has been a list leading to the fact that the checkboxes were mistaken as bullet points instead of selectable checkboxes.

Now, the logging console is displayed as a table in which the checkboxes are displayed in a separate column being easier to recognize by the user.

Apart from that, the logged statements get highlighted when hovering over them, implying that these items are clickable. With this improvement, the user now sees that there is another feature that appears when clicking on one of the statements.

Both participants appreciated the changes and agreed that the implementation was a satisfactory outcome.

6.1.5 Button Improvements

The behavior of the reset button has been rather unclear during the first evaluation. When trying to reset a monitoring session, a dialog pops up asking the user which filters he wants to keep, but this was formulated in a difficult way.

After reformulating this text, both participants agreed that the different types of filters are now defined in a clear and structured way and are therefore comprehensible for the user.

Apart from that, a download button has been added to the login view of LiMoxAPI providing the possibility of downloading the login credentials of the used LRS. This has also been received as a useful feature, although one of the participants noted that an import feature should also be provided in order to reload the downloaded credentials back into the application.

6.2 Overall Results

In general, both participants seemed quite content with the implementation of their suggestions from the user evaluation. This study has shown that there is a great interest in LiMoxAPI, not only coming from VR developers which use the OmiLAXR Ecosystem, but also from LA developers from other fields.

Conducting an evaluation with participants from different application fields has led to a broader perspective on the analysis of xAPI statements and therefore made a massive difference in the enhancement of LiMoxAPI as well as confirming the profit of the application in the support of a developers' verification process of learning data. There is still some potential in LiMoxAPI which will be further discussed in the chapter Future Work, but the applied changes already achieve a better user experience as well as a better chance of verifying and analyzing xAPI statements.

The overall application provides a simpler usage due to additional explanatory texts and user feedback at the occurrence of errors. Due to the created interplay of the different components and the enhanced filter complexity, the user gains an improved data integrity and thus the ability of a more sophisticated analysis.

Sometimes, it is a difficult trade off between enhancing the usability and data integrity in LiMoxAPI while ensuring that the application stays a lightweight and sophisticated live monitoring tool, yet the added changes were necessary in order to provide a proper monitoring tool for LA developers.

Chapter 7 Future Work

During both evaluation processes and the implementation of the suggested enhancements, a lot of ideas for LiMoxAPI arose and have been discussed. Both participants individually had a great impact on the further development of the application, but not every feature could be implemented due to time constraints and changing demands. Especially after the final evaluation, there was not much time left to develop further changes.

This section will sum up particular options for improving LiMoxAPI with concrete plans about their implementation.

7.1 Enhancing Existing Features

In the final evaluation, one participant noted that it would make more sense to transform the developed history log from the pulse monitor into a histogram instead of portraying the data in a list.

In order to achieve this it would make sense to restructure the whole pulse monitor component by replacing the ApexCharts library with a different library for example by using Chart.js¹ or Apache's ECharts². Enhancing the pulse monitor in general has shown to be quite tedious because ApexCharts turns out to not be that performant when dealing with large datasets, especially in real-time scenarios which exist in LiMoxAPI [Per].

Charts.js is lightweight, has good performance and is easy to use, ECharts is highly performant, can handle large datasets and supports real-time updates [Ges21].

Both libraries are also supported by Vue and thus are easy to integrate. Due to time issues this could not be performed in the scope of this project, but the styling and displaying options of the chart have been adjusted such that there is no visible performance issue when using the application.

The enhancements of the filter component have achieved a great difference in the usage of the overall application. One thing that could be added here would be a feature which allows to save created filters with the option to import them for the next monitoring session. This way, users are able to reuse filters and their provided logic which would save effort and time.

A feature which correlates to that is the download button for the login credentials. When providing the option to download the entered login credentials, LiMoxAPI should also enable the user to import them such that they do not have to copy them

¹ <https://www.chartjs.org/docs/latest/>, accessed: February 2025

² <https://echarts.apache.org/handbook/en/get-started/>, accessed: February 2025

from their `credentials.json` file, but rather are able to just select the correct credentials file in their browser leading to an automatically filled out login form.

7.2 Changes in LiMoxAPI's Architecture

This section describes possible future changes in LiMoxAPI regarding the construction of its architecture and will look at three possibilities.

7.2.1 Integration of the YetAnalytics SQL LRS

Currently, LiMoxAPI connects to an external LRS which either runs locally on the computer of the user, or runs on the web like LL.

This LRS receives the generated learning data from the learning environment that should be monitored. For an easy monitoring process, it is recommended to use the pre configured YetAnalytics SQL LRS because it can be run locally and does not require an additional configuration. When using this LRS, it has to be started each time when LiMoxAPI gets started.

Thus a great improvement would be to integrate the LRS into LiMoxAPI. One realization of this could be done by adding the pre configured LRS to the LiMoxAPI package. In addition, the according launch functionality has to be added to the rust backend. This would provide a rather simple way to integrate an LRS into LiMoxAPI. But it is still important to give the user the option to connect to a different LRS such that users are not forced to use the SQL LRS from YetAnalytics.

7.2.2 Developing a Plugin System

One aim of this project has been to make LiMoxAPI accessible to a wider range of users. This could be realized by providing a plugin system such that users can create their own custom features.

During this project the demand for this idea was not high enough and therefore the focus rather shifted towards enhancing the already existing components.

Still, one way to realize such a plugin system would be using Rust traits³. One could define a feature trait with the required functions to process the passed data and send it to the frontend with the use of the Tauri communication system. There, an according event-listener is needed which acts upon the arrival of the sent data.

Nonetheless, the evaluation has shown that LA developers do not want to learn the Rust programming language in order to create small features. Most of them would only use this option for research purposes rather than for an analysis and verification purpose.

Thus, users would only use this feature occasionally and if they would, they want to have a simple way to implement their ideas.

Another option would be to use JS Sandbox⁴. *js-sandbox* is a Rust library which enables the execution of JS code in Rust in a secure sandbox.

However, one of the participants in the user study stressed that this would be quite tedious and time-consuming. He suggested implementing a simple RestAPI such that

³ <https://doc.rust-lang.org/book/ch10-02-traits.html>, accessed: February 2025

⁴ <https://crates.io/crates/js-sandbox>, accessed: February 2025

developers can write easy Python code. Still, this solution has to be compatible with the Tauri structure of LiMoxAPI.

7.2.3 Creating a REST-API

In the last few weeks of the thesis, the idea came up to dockerize LiMoxAPI instead of creating a binary executable, thus making the application more modular. Two docker images would be needed for this, one for the backend and one for the frontend (see Figure 7.1)⁵.

This way, LiMoxAPI would not just stay a desktop application, but rather get the option to be run as a Web App.

In order to achieve this, the Tauri IPC has to get disconnected from the application and has to get replaced by a REST-API with defined HTTP routes and JSON data handling.

One good option to achieve this is via the web framework Actix Web [cona]. It is a powerful and fast web framework in Rust which allows the development of web applications and APIs.

The *Commands* in the backend can be called via REST API routes whose endpoints are defined in `main.rs`. By creating an API service in the frontend which will replace the invoke handlers to the backend, it will be possible to make API calls and send HTTP requests to the backend by using the Axios library [conb] which is a Promise based HTTP client for the browser and Node.js.

In order to realize emissions from the backend to the frontend meaning *Events* in the Tauri IPC, one could make use of WebSockets which are also provided by Actix Web allowing bidirectional data passing.

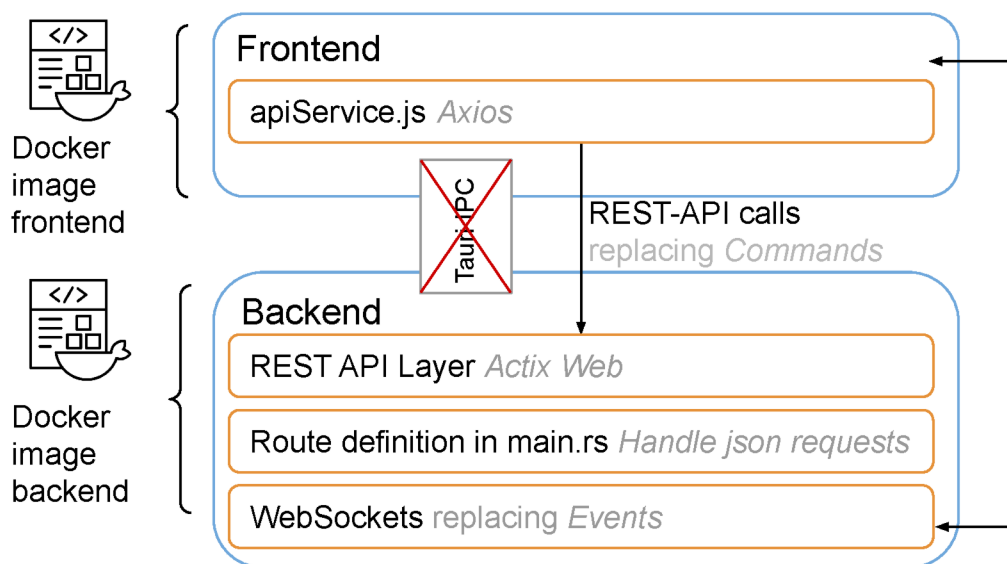


Figure 7.1: Possible Docker Architecture

⁵ This figure uses resources from <https://www.flaticon.com/>

Chapter 8 Conclusion

The main goal of this project was to find out how the usability and data integrity in the existing version of LiMoxAPI could be enhanced in order to provide an even better live monitoring tool for LA developers.

Originally, the application was a prior bachelor thesis which has been developed in the context of the OmiLAXR Ecosystem provided by the research group Learning Technologies.

Therefore, it was mainly provided for LA developers with a VR context.

It was aimed to broaden the application field of LiMoxAPI such that LA developers from all fields were able to engage with the application.

In order to achieve this, a user evaluation has been conducted with two participants from different LA application fields, widening the perspective on the analysis of xAPI statements.

Initially, LiMoxAPI was a solid prototype for monitoring incoming learning data in a connected LRS but there existed multiple issues. The application was stiff, all components were independent from each other and did not provide any interplay.

Users also did not have many options to interact with the desktop application and thus were limited in their analysis potential.

The conducted user evaluation led to the retrieval of multiple different suggestions for additional features as well as further improvements for existing components.

The main focus lay on the overall simpler understanding of LiMoxAPI thus an improved user experience.

In addition, this thesis strived for a more complex interplay between the different components and more options for the user to engage with the application with a particular focus on the filter component.

This component had the biggest changes, being the enablement for RegEx search in the value field of a filter rule and the option to provide filter logic between different filter rules.

In addition, users are now able to analyze single aspects more intensely and individually due to the ability of logging the filtered statements of all filters in the logging console or only logging the filtered statements of a specific filter in a separate filter-log.

Moreover, users are able to either reset the whole monitoring session while keeping specific filters or to only reset the statement counter of a specific filter.

The other components obtained a restructuring by adding x-axis labels to the pulse monitor and displaying the logging console as a table, because it now contains checkboxes which enable the user to select a logged statement as a filter.

Besides that, additional coherent explanatory details and graph definitions led to a

simpler comprehension of the given components, their interplay and therefore the overall application.

Due to these implemented changes, users are now able to monitor, analyze and thus verify their xAPI data in a simpler way.

This also sets the path for an improved data integrity due to enhanced feedback to the user and an overall better clarification of the application reducing user errors and incorrect actions but rather enhancing analysis potential.

Nevertheless, the final evaluation has shown that there still exist possible improvements for LiMoxAPI which could not be realized in the scope of this project due to time constraints.

These contain the restructuring of the pulse monitor by replacing the ApexCharts library with a more performant library, for example being Chart.js or Apache's ECharts. In addition, some little adjustments in the frontend components would be helpful. Bigger changes would be the realization of a plugin system, as well as the inclusion of the YetAnalytics LRS to automate its starting process.

Finally, the code of LiMoxAPI could be restructured such that the Tauri Framework gets replaced by an HTTP REST-API such that its backend and frontend can be dockerized individually.

All in all, the goal of enhancing the usability and data integrity of LiMoxAPI was achieved resulting in a performant and sophisticated live monitoring tool for LA developers of all application fields. There is still room for improvement and enhancement, but it is always a trade off between upgrading LiMoxAPI while ensuring that the application stays a lightweight live monitoring tool.

Appendix A Bibliography

- [BKP⁺16] Aneesha Bakharia, Kirsty Kitto, Abelardo Pardo, Dragan Gašević, and Shane Dawson. Recipe for success: lessons learnt from using xapi within the connected learning analytics toolkit. In *Proceedings of the sixth international conference on learning analytics & knowledge*, pages 378–382, 2016.
- [BT75] Victor R Basil and Albert J Turner. Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering*, (4):390–396, 1975.
- [CDST12] Mohamed Amine Chatti, Anna Lea Dyckhoff, Ulrik Schroeder, and Hendrik Thüs. A reference model for learning analytics. *International journal of Technology Enhanced learning*, 4(5-6):318–331, 2012.
- [Clo13] Doug Clow. An overview of learning analytics. *Teaching in Higher Education*, 18(6):683–695, 2013.
- [cona] Actix contributors. Actix web documentation. <https://actix.rs/docs/>, Accessed: February 2025.
- [conb] Axios contributors. Axios documentation - introduction. <https://axios-http.com/docs/intro>, Accessed: February 2025.
- [CV97] Mahil Carr and June Verner. Prototyping and software development approaches. *Department of Information Systems, City University of Hong Kong, Hong Kong*, pages 319–338, 1997.
- [For18] Jodi Forlizzi. Moving beyond user-centered design. *interactions*, 25(5):22–23, 2018.
- [Fri06] Jeffrey Friedl. *Mastering regular expressions*. " O’Reilly Media, Inc.", 2006.
- [Ges21] Ralf Geschke. Javascript chart libraries im benchmark: Apexcharts, chart.js, apache echarts, 2021. <https://www.kuerbis.org/2021/02/javascript-chart-libraries-im-benchmark-apexcharts-chart-js-apache-echarts/>, 2021, Accessed: February 2025.
- [GHS24] Sergej Görzen, Birte Heinemann, and Ulrik Schroeder. Towards using the xapi specification for learning analytics in virtual reality. In *LAK Workshops*, pages 260–271, 2024.

- [GTE⁺17] Maria Grandl, Behnam Taraghi, Markus Ebner, Philipp Leitner, and Martin Ebner. Learning analytics. *Handbuch E-Learning. Expertenwissen aus Wissenschaft und Praxis-Strategien, Instrumente, Fallstudien*. Köln: Wolters Kluwer, pages 1–16, 2017.
- [Gö24] Sergej Görzen. Omilaxr, 2024. <https://omilaxr.dev/get-started/>, 2024, Accessed: February 21, 2025.
- [Ham23] Nasrullah Hamidli. Introduction to ui/ux design: key concepts and principles. *Academia*. URL: https://www.academia.edu/98036432/Introduction_to_UI_UX_Design_Key_Concepts_and_Principles [accessed 2024-04-27], 2023.
- [HEGS22] Birte Heinemann, Matthias Ehlenz, Sergej Görzen, and Ulrik Schroeder. xapi made easy: A learning analytics infrastructure for interdisciplinary projects. *International Journal of Online & Biomedical Engineering*, 18(14), 2022.
- [HGD⁺24] Birte Heinemann, Sergej Görzen, Ana Dragoljic, Lars Meiendresch, Marc Troll, and Ulrik Schroeder. A learning analytics dashboard to investigate the influence of interaction in a vr learning application. In *LAK Workshops*, pages 251–259, 2024.
- [HGS⁺22] Birte Heinemann, Sergej Görzen, Ulrik Schroeder, J Bourdin, and E Paquette. Repix vr-learning environment for the rendering pipeline in virtual reality. *Bourdin, J.-J.; Paquette, E., Hrsg.): Eurographics*, 2022.
- [HGS23] Birte Heinemann, Sergej Görzen, and Ulrik Schroeder. Teaching the basics of computer graphics in virtual reality. *Computers & Graphics*, 112:1–12, 2023.
- [Hol93] Andy Holyer. Methods for evaluating user interfaces. 1993.
- [JDT20] Akhila Joshi, Padmashree Desai, and Prakash Tewari. Learning analytics framework for measuring students’ performance and teachers’ involvement through problem based learning in engineering education. *Procedia Computer Science*, 172:954–959, 2020.
- [KE15] Mohammad Khalil and Martin Ebner. Learning analytics: principles and constraints. In *Edmedia+ innovate learning*, pages 1789–1799. Association for the Advancement of Computing in Education (AACE), 2015.
- [LRV⁺08] Effie Law, Virpi Roto, Arnold POS Vermeeren, Joke Kort, and Marc Hassenzahl. Towards a shared definition of user experience. In *CHI’08 extended abstracts on Human factors in computing systems*, pages 2395–2398. 2008.
- [MG19] Katerina Mangaroska and Michail Giannakos. Learning analytics for learning design: A systematic literature review of analytics-driven design to enhance learning. *IEEE Transactions on Learning Technologies*, 12(4):516–534, 2019.

- [MHH18] Angela Minichiello, Joel R Hood, and Derrick Shawn Harkness. Bringing user experience design to bear on stem education: A narrative literature review. *Journal for STEM Education Research*, 1:7–33, 2018.
- [Per] Robin Percy. Comparing the most popular javascript charting libraries. <https://blog.logrocket.com/comparing-most-popular-javascript-charting-libraries/>, 2023, Accessed: February 2025.
- [Rad14] Iulian Radu. Augmented reality in education: a meta-review and cross-media analysis. *Personal and ubiquitous computing*, 18:1533–1543, 2014.
- [RSS21] Jenny Ruiz, Estefanía Serral, and Monique Snoeck. Unifying functional user interface design principles. *International Journal of Human–Computer Interaction*, 37(1):47–67, 2021.
- [Sie12] George Siemens. Learning analytics: envisioning a research discipline and a domain of practice. In *Proceedings of the 2nd international conference on learning analytics and knowledge*, pages 4–8, 2012.
- [SLMOH⁺17] Ángel Serrano-Laguna, Iván Martínez-Ortiz, Jason Haag, Damon Regan, Andy Johnson, and Baltasar Fernández-Manjón. Applying standards to systematize learning analytics in serious games. *Computer Standards & Interfaces*, 50:116–123, 2017.
- [TSL24] Georg Thiesen, Ulrik Schroeder, and Horst Lichter. A Learning Analytics Monitoring Tool for Supporting VR Developers, 2024.
- [Väk23] Valtteri Väkevä. Tracking learning experiences with xapi. Master’s thesis, 2023.
- [VODC⁺20] Katrien Verbert, Xavier Ochoa, Robin De Croon, Raphael A Dourado, and Tinne De Laet. Learning analytics dashboards: The past, the present and the future. In *Proceedings of the tenth international conference on learning analytics & knowledge*, pages 35–40, 2020.
- [VRL15] Juan C Vidal, Thomas Rabelo, and Manuel Lama. Semantic description of the experience api specification. In *2015 IEEE 15th International Conference on Advanced Learning Technologies*, pages 268–269. IEEE, 2015.
- [WKS22] Salome Wörner, Jochen Kuhn, and Katharina Scheiter. The best of two worlds: A systematic review on combining real and virtual experiments in science education. *Review of Educational Research*, 92(6):911–952, 2022.

Appendix B User Evaluation Script

This appendix contains the script which was followed while conducting the first round of user evaluations.

It holds the motivation for LiMoxAPI, the five different tasks the participants had to solve as well as the questions which were discussed after each task and the further ideas that have been presented and discussed at the end of the study.

The evaluation itself was supported by presentation slides which have been created from the given script.

The script is displayed on the next two pages.

Erste Nutzer-Evaluierung von LiMoxAPI

Alle Teilnehmenden erhalten vor der Evaluation die neueste Version von LiMoxAPI und sollen prüfen, ob sich die Anwendung öffnen lässt, man sich verbinden kann (mit der YetAnalytics LRS, die lokal auf dem Rechner läuft) und man auf die Dashboard-Seite von LiMoxAPI gelangt.

Anschließend wird die Evaluation in einem Zoomraum durchgeführt, wichtige Informationen und Grafiken werden dabei anhand von PowerPoint Folien gezeigt, und der Ablauf orientiert sich dabei an der folgenden Tabelle:

Begrüßung	<ul style="list-style-type: none">• Herzlich Willkommen• Danke für Interesse an meinem Projekt und Bereitschaft zur Teilnahme• Lea, 7. Semester vom Informatikstudium, Bachelorarbeit i9• In dieser Studie soll die Desktopanwendung LiMoxAPI getestet werden• Ich werde Notizen machen zur Vorgehensweise aller Teilnehmenden• es gibt keine richtige oder falsche Verhaltensweise, nur das Tool wird bewertet und nicht die Aktionen der Teilnehmenden
Einführung ins Thema (Thema kurz vorstellen, Bildschirm teilen, Grafik zeigen, die Zusammenhang mit OmiLAXR Ecosystem und LiMoxAPI erklärt, LiMoxAPI zeigen, Fragen zur Motivation beantworten)	<ul style="list-style-type: none">• LiMoxAPI = Live Monitoring of xAPI Statements• Soll den Entwicklungsprozess von Lernumgebungen erleichtern• Ursprünglich für VR-Entwicklung gedacht, doch Interesse kam auch in anderen Bereichen auf• Erweitert das OmiLAXR Ecosystem vom Lehrstuhl für Lerntechnologien• Soll ein simples, lightweight tool neben Learning Locker sein → greift die xAPI-Statements vom LRS ab und stellt sie grafisch dar (Verifizierung der Generierung von Lerndaten)• Läuft lokal neben der Lernumgebung• Nutzt den YetAnalytics LRS um Daten abzugreifen (läuft lokal auf dem Rechner)• LiMoxAPI ist eine Tauri App, welche im Frontend mit Vue.js und im Backend mit Rust läuft
1. Aufgabe (Vorstellungen zu den einzelnen Komponenten notieren, anschließend Funktionalität von Komponenten erklären)	Verbinden Sie sich mit dem LRS von YetAnalytics. Nun kommen Sie zu dem LiMoxAPI Dashboard. Was denken Sie was die einzelnen Komponenten machen?
2. Aufgabe (Es wird ein Beispiel xAPI Statement angezeigt. Erwähnen, dass xAPI Statements (wie im gesamten OmiLAXR Ecosystem) nach dem TinCan Standard definiert sind)	Erstellen Sie einen Filter, der zählt wie oft eine Maus von dem Nutzer benutzt wurde und lassen Sie nur die gefilterten Statements in der Logging Console anzeigen (Lösung: path: object.definition.name.en-US, value: mouse)

3. Aufgabe (Ist eventuell etwas unintuitiv, nachhelfen falls keiner darauf kommt)	Lassen Sie sich ein xAPI statement in komplettem JSON-Format anzeigen
4. Aufgabe	Starten Sie die Monitoring-Session neu und wählen Sie ein x-API Statement aus der Logging Console aus, welches als Filter übernommen werden soll. Stellen Sie dann manuell ein, dass Sound 3 ertönen soll, wenn das Statement NICHT vorkommt.
5. Aufgabe Anschlussfrage stellen: Wie nützlich ist dieses Pop-up Fenster? (Kommt nur, wenn Session resetted wird und noch dynamisch hinzugefügte Filter existieren)	Probieren Sie ein bisschen mit dem Reset Button und den Filtern aus der Logging Console aus und bewerten Sie die Funktion des pop-up Fensters beim Reset.
Feedbackrunde (Antworten notieren, nachfragen wenn etwas nicht ganz klar ist)	<ul style="list-style-type: none"> • Was fanden Sie bei der Nutzung unintuitiv? • Wie lässt sich das Anzeigen von ganzen Statements in JSON-Format intuitiver darstellen? • Wie lässt sich die Bedienung des Filters verständlicher darstellen? • Was fiel Ihnen schwer oder war unklar? • Was vermissen Sie an Funktionalität in LiMoxAPI? • Wenn Sie ein Feature weglassen müssten, welches wäre das?
Vorstellung weiterer Pläne (Nach jedem Punkt diskutieren und Meinungen der Teilnehmer einholen)	<ul style="list-style-type: none"> • Download Button um Einlog-Credentials vom LRS lokal zu speichern • Vernetzung von Statements bei den Filtern ermöglichen (zurzeit nicht viel Logik dahinter) (Wie können Sie sich vorstellen das zu realisieren? Beim Konfigurieren eines Filters nach Filterlogik gefragt werden? Verunden, Verodern, Verneinen, ...) • RegEx bei den Filtern ermöglichen • Dropdown Menü mit Statement Vorschlägen, um Beispielstruktur zu zeigen bzw. stattdessen sogar Auto-Completion bei der Filterfunktion, die Werte zum filtern vorschlägt (dafür wäre eingebundene Datenbank notwendig) • Button um Session zu pausieren (pausierte Sessions sollen farbig gekennzeichnet sein, damit der Nutzer jederzeit weiß, dass gerade keine Monitoring Session läuft) • Plugin System (Nutzer können ihre eigenen kleinen Features erstellen bevor die Anwendung gebaut wird: Meisten Aktivitäten eines Nutzers anzeigen, ...)
Abschließende Fragen	<ul style="list-style-type: none"> • Welche Features finden Sie für Ihren Bereich notwendig/interessant? • Würde sich für Sie ein solches Plugin-System lohnen? • Wie könnte die Anwendung intuitiver werden, bzw. wie kann die User Experience gesteigert werden? • Haben Sie für das Design konkrete Verbesserungsvorschläge?

Glossary

JSON	The JavaScript Object Notation is a common data format to store and transmit data.
LL	Learning Locker is the Learning Record Store which is used in combination with the OmiLAXR Ecosystem.
RegEx	A regular expression is a pattern which describes a set of strings that matches the pattern.
SQL LRS	the Learning Record Store of YetAnalytics.

Acronyms

HMD	Head Mounted Display.
IPC	Inter-Process Communication.
LA	Learning Analytics.
LiMoxAPI	Live Monitoring of xAPI Statements.
LMS	Learning Management System.
LRS	Learning Record Store.
OmiLAXR	Open and modular integration of Learning Analytics in eXtended Reality.
UI	User Interface.
UX	User Experience.
xAPI	experience API.
XR	eXtended Reality.

Eidesstattliche Versicherung

Declaration of Academic Integrity

Sieler, Lea

Name, Vorname/Last Name, First Name

434459Matrikelnummer (freiwillige Angabe)
Student ID Number (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit~~/Bachelorarbeit/
Masterarbeit* mit dem Titel

I hereby declare under penalty of perjury that I have completed the present paper/bachelor's thesis/master's thesis* entitled

Enhancing Usability and Data Integrity in LiMoxAPI for LA Developers

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt; dies umfasst insbesondere auch Software und Dienste zur Sprach-, Text- und Medienproduktion. Ich erkläre, dass für den Fall, dass die Arbeit in unterschiedlichen Formen eingereicht wird (z.B. elektronisch, gedruckt, geplottet, auf einem Datenträger) alle eingereichten Versionen vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

independently and without unauthorized assistance from third parties (in particular academic ghostwriting). I have not used any other sources or aids than those indicated; this includes in particular software and services for language, text, and media production. In the event that the work is submitted in different formats (e.g. electronically, printed, plotted, on a data carrier), I declare that all the submitted versions are fully identical. I have not previously submitted this work, either in the same or a similar form to an examination body.

Aachen, February 22, 2025

Ort, Datum/City, Date

L. Sieler

Unterschrift/Signature

*Nichtzutreffendes bitte streichen/Please delete as appropriate

Belehrung:

Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 156 StGB (German Criminal Code): False Unsworn Declarations

Whosoever before a public authority competent to administer unsworn declarations (including Declarations of Academic Integrity) falsely submits such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment for a term not exceeding three years or to a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Strafflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtet. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

§ 161 StGB (German Criminal Code): False Unsworn Declarations Due to Negligence

(1) If an individual commits one of the offenses listed in §§ 154 to 156 due to negligence, they are liable to imprisonment for a term not exceeding one year or to a fine.

(2) The offender shall be exempt from liability if they correct their false testimony in time. The provisions of § 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

I have read and understood the above official notification:

Aachen, February 22, 2025

Ort, Datum/City, Date

L. Sieler

Unterschrift/Signature

Acknowledgement

I want to thank my supervisors, Sergej Görzen and Sarah Sahabi, for their support and feedback throughout my thesis. Besides, I want to thank the two participants whose participation allowed me to conduct the user evaluations and helped to improve LiMoxAPI. At last, I am thanking the research group Learning Technologies for giving me the opportunity to write my bachelor thesis.