

The present work was submitted to Lehr- und Forschungsgebiet Educational Technologies at DIPF

Building a connector for Moodle

Bachelor-Thesis

Presented by

Schröber, Jakob

ID: 3813605

First examiner: Prof. Dr. Hendrik Drachsler

Supervisor: George-Petru Ciordas-Hertel

Frankfurt, September 22, 2020

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Ebenso bestätige ich, dass diese Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed. Furthermore, I confirm that neither this work as a whole nor part of it were used before for any another exam or assignment.

Frankfurt, September 22, 2020

Jakob Schröber

Acknowledgments

My deepest gratitude goes to my family. I would especially like to thank my father and mother for paving the way to my academic success and for being such loving parents. I sincerely thank my wife Delia who has always been supportive throughout my studies as well as the last months that I have been working on this project.

I would like to thank my first examiner Prof. Hendrik Drachsler for accepting my thesis project and providing very beneficial feedback in response to my initial and interim presentation.

I want to use the chance to explicitly thank my supervisor George Ciordas-Hertel for promoting this thesis project with his many helpful thoughts and considerations. I want to point out that he was a very professional but at the same time inspiring mentor. His feedback was always constructive, and I always felt supported and encouraged by his comments.

I also want to thank the second examiner for taking the time and effort to review this thesis. Unfortunately, I can't address him or her personally as the examination office decides who will be the one.

Finally, I want to express my gratitude to all my friends and colleagues of the degree program of Computer Science at Goethe University. I had a great time studying with you. Take care, I hope we will see each other again.

Abstract

This thesis reports the development of an application built to assess learners in the learning platform Moodle based on Moodle's log-file data. In comparison to existing Learning Analytics applications for Moodle, it combines the flexibility to use all data that Moodle collects in its database with the comfort of a scheduled assessment procedure. For any possible construct, the application allows for the individual design of indicators in order to measure that construct. In order to obtain the assessment results for a specific construct, values for each indicator are calculated and then preprocessed according to a previously chosen scaling method. According to previously set construct-specific indicator weights, a weighted average is formed for each learner in each course based on his or her indicator's values. Following the notion of Ongoing Assessment, construct assessments can be scheduled in order to gather evidence over time and across contexts. The results of these assessments are provided in two ways, aiming at improved interoperability: The latest results can be obtained in JSON format by calling a REST API. Additionally, historical results are exported to a NoSQL database. Eventually, the application is evaluated by exemplarily implementing scheduled assessments of the construct of Learning Engagement. Applied in the contexts of Learning Analytics and Digital Formative Assessment, the results of scheduled Learning Engagement assessments could be used to identify students at risk, to improve teaching strategies and to support self-regulated learning.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Research Questions	2
1.3	Outline	2
2	Background	3
2.1	The construct of Learning Engagement	3
2.2	Ongoing and Digital Formative Assessment	5
2.3	Learning Analytics research	6
2.4	The challenge of interoperability	7
3	Related Work	9
3.1	Acquisition of log-file data from Moodle	9
3.1.1	Database query	9
3.1.2	Moodle core functionality	9
3.1.3	Moodle plugins and blocks	10
3.1.4	External applications	11
3.1.5	Summary	12
3.2	Learning Engagement detection using log-file data	13
3.3	Contribution of this project	19
4	Concept and Design	21
4.1	Epics	21
4.2	Features	21
4.2.1	Pending assessments identification	22
4.2.2	Assessment scheduling	22
4.2.3	Assessment performance	22
4.2.4	Results storage	22
4.2.5	Results report	23
4.2.6	Results emission	23
4.3	Acceptance criteria	23
4.4	Design objectives and decisions	25
4.4.1	Flexibility	25
4.4.2	Extensibility	25
4.4.3	Reusability	25
4.4.4	Modularity	26
4.4.5	Compatibility	27

5	Implementation	29
5.1	Technologies	29
5.1.1	Docker	29
5.1.2	Python	29
5.1.3	Django	30
5.1.4	Scikit-learn	31
5.1.5	Celery	31
5.2	Project and apps	32
5.2.1	'app' project	32
5.2.2	'data_source' app	33
5.2.3	'administration' app	33
5.2.4	'assessment' app	34
5.2.5	'export' app	42
5.3	Project workflow	44
6	Evaluation	47
6.1	Reconsidering features and acceptance criteria	47
6.2	Assessment of Learning Engagement (User guide)	51
6.2.1	Software installation, system startup and provision of courses and users	51
6.2.2	Activation of assessment for courses and their users to the project	52
6.2.3	Configuration of Learning Engagement indicators	53
6.2.4	Configuration of the Learning Engagement construct	56
6.2.5	Assignment of weights to the indicators	58
6.2.6	Scheduling of assessments	60
6.2.7	Analysis of results	61
6.3	Review and limitations	64
7	Discussion	65
7.1	Conclusion	65
7.2	Outlook	66
	List of Tables	67
	List of Figures	69
	Bibliography	71
	Appendices	75

1 Introduction

In this chapter, I motivate my research by introducing to the topic and present my research questions. At the end of the chapter, I outline the subsequent chapters of the thesis.

1.1 Motivation

Computers and information systems have been used for many years now in almost every field, often gathering huge amounts of data of potential value for the respective area. This inspires attempts to apply big data technologies to non-private, sovereign fields like health or security. The opportunities that new technologies create for these often highly institutionalized areas stand opposite the effects that potentially arise from their adoption.

One of the societal fields facing this kind of attempts is education. When educational institutions only slowly adopt new technologies, questions arise whether schools still prepare young people adequately for the challenges of the 21st century (Shute et al., 2016). On the other hand, it becomes more difficult in learning environments that no longer require the students' physical attendance to ensure that students are actually engaged with the learning materials.

Notwithstanding the use of new technologies in education, disengagement has become a problem in many developed countries' education systems. There is evidence that half of the students feel bored in school every day, ten percent even hate school (Shum & Crick, 2012). This is a severe problem considering that research has linked high engagement to satisfaction, persistence, achievement and the acquisition of problem-solving skills (Yang et al., 2018).

Moreover, closed schools and universities during the COVID-19 pandemic have most likely increased the learning gap between children from lower-income and higher-income families. Non-school factors are seen as a primary source of inequalities in educational outcomes (Van Lancker & Parolin, 2020). In this situation, a scheduled assessment of students' engagement levels could have provided a more objective measure and evidence for the anticipated effects.

But even if schools are open again, the share of online learning experiences will rather grow. Schools and universities will increasingly use the opportunities of learning platforms such as Moodle. Those learning platforms typically capture their usage in form of log-file data, which could be used in scheduled assessments of students' levels of engagement with the system.

This thesis provides a flexible solution for scheduled assessments of students' engagement. Scheduled assessment of students' engagement levels could serve the following purposes: First, students at risk could be identified and receive help before their disengagement manifests in declining grades and dropout. Second, the teaching itself could benefit by adapting and personalizing teaching strategies to students' different levels of engagement. Third, a monitoring of their engagement levels could benefit older students in support of self-regulated learning.

1.2 Research Questions

The learning platform Moodle is free of charge open source and very popular. To my knowledge, no existing application allows the flexible configuration of scheduled assessments of students' engagement based on Moodle's log-file data. What's more, to my knowledge no application allows the flexible configuration of scheduled assessments of an arbitrary number of constructs. Consequently, it is the objective of this thesis to implement such a flexible solution for Moodle. My research questions are the following:

1. How can flexibly configurable constructs be assessed using Moodle's log-file data?

As a latent construct, it is not possible to measure engagement or a similar construct directly; instead, those constructs must be measured by a number of well-defined indicators. I believe that a solution that can be used for more than one construct is more beneficial than a single-construct solution and therefore I will present a way to assess a flexibly configurable construct. This will require the opportunity to define multiple indicators that leverage Moodle's log-file data. The solution will aggregate the indicators' values into a single value for the chosen construct.

2. How can construct assessments be scheduled in order to obtain the results regularly?

The solution will be especially helpful if the assessments of the configured constructs can be carried out automatically over the long term. This way, the development of the values over time will become visible, for individuals as well as for groups of learners. In order to carry out such an automated assessment, a scheduling mechanism is needed that triggers the assessment repeatedly at certain points. The solution will implement such a flexible scheduling mechanism.

3. How can interoperability be achieved for the results of the construct assessments?

In order to access the results of the scheduled construct assessments at a later stage, their results will have to be persisted. Interoperability increases the probability that other applications will be able to more or less directly use the results of the scheduled construct assessments. The solution will provide the results of the scheduled construct assessments in a preferably interoperable way.

1.3 Outline

The remaining part of the thesis is organized as follows: In the second chapter, I introduce to the construct of Learning Engagement and review the main concepts and fields of study that are important for this thesis. The third chapter covers the related work regarding the acquisition of log-file data from Moodle and the detection of Learning Engagement using log-file data. It ends with the contribution of this thesis. I then proceed with the fourth chapter which describes the concept and design of the proposed implementation, including design principles and acceptance criteria. The fifth chapter describes the implementation. In the sixth chapter, I evaluate the solution by checking against the acceptance criteria and by exemplarily implementing scheduled assessments of the construct of Learning Engagement. Finally in the seventh chapter, I draw the conclusion and present an outlook.

2 Background

This chapter gives a brief background about the concepts and fields of study relevant for this thesis. It starts with a description of the concept of Learning Engagement and methods of its measurement. It progresses with an introduction to the vision of Ongoing Assessment in education and the promises of Digital Formative Assessment. The third section of this chapter gives an overview about the research field of Learning Analytics and learning activity data. The fourth section explains the challenge of interoperability in the context of Learning Analytics.

2.1 The construct of Learning Engagement

In the European Union, still more than 10% of students leave school too early (European Union, 2019). The construct of Learning Engagement (LE) is relevant in this context. Low engagement has been linked to negative learning outcomes (Pardos et al., 2013) and is considered one of the most relevant factors for school dropout (Aluja-Banet et al., 2019). Yang et al. (2018) report that, in online contexts, the following definition is most often used for Learning Engagement:

Engagement: the time and energy students devote to educationally sound activities inside and outside of the classroom, and the policies and practices that institutions use to induce students to take part in these activities (Kuh, 2003).

Yet, this definition focuses only on externally observable behavior. Alternative definitions have already early established the idea that Learning Engagement is a multifaceted construct. For example, Fredricks et al. (2004) identified two additional conceptualizations of LE besides the already established behavioral component:

Behavioral engagement draws on the idea of participation; it includes involvement in academic and social or extracurricular activities and is considered crucial for achieving positive academic outcomes and preventing dropping out. Emotional engagement encompasses positive and negative reactions to teachers, classmates, academics, and school and is presumed to create ties to an institution and influence willingness to do the work. Finally, cognitive engagement draws on the idea of investment; it incorporates thoughtfulness and willingness to exert the effort necessary to comprehend complex ideas and master difficult skills (Fredricks et al., 2004).

Similarly, Bosch (2016) distinguishes three components of LE: affective, cognitive and behavioral engagement. While he makes a clear distinction between the methods of annotation which he sorts into the categories of self-reports and third-party observations, he emphasizes that progress has been made in the machine-based detection of affective and cognitive states based on external cues.

Dewan et al. (2019) have developed a taxonomy of methods of LE detection, which is shown in Figure 2.1. They divide the existing methods into three main categories: manual, semi-automatic and automatic methods of detection.

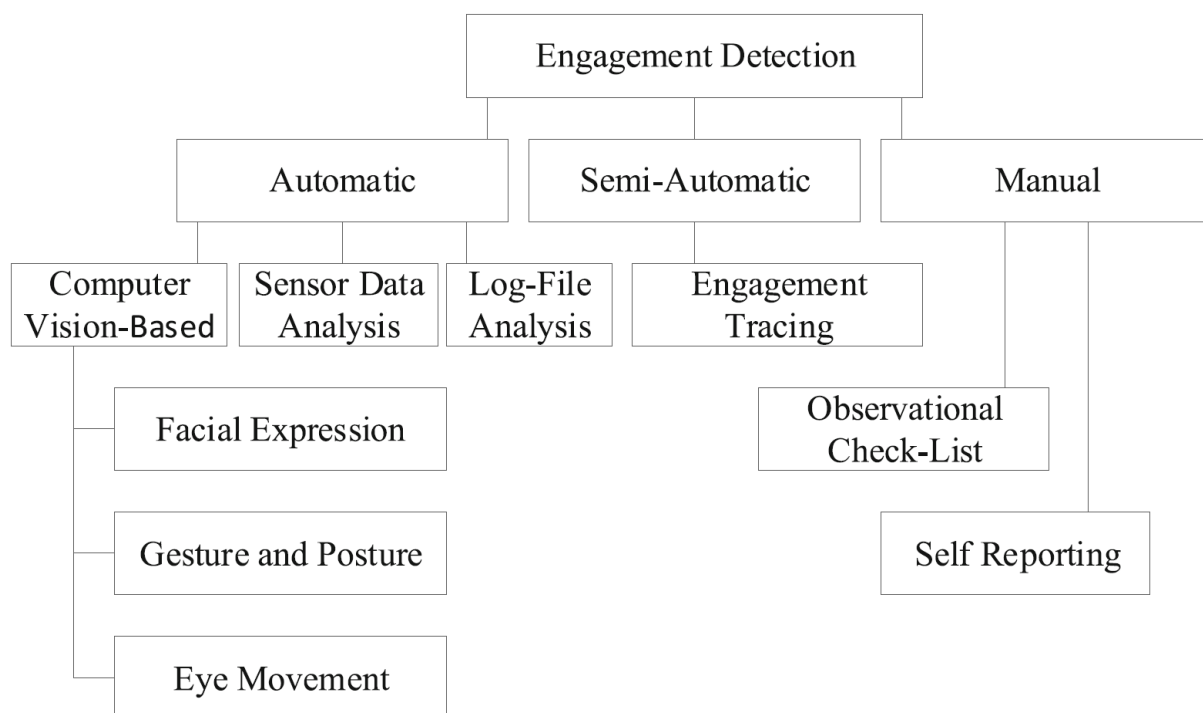


Figure 2.1: Learning Engagement detection methods

Source: adopted from Dewan et al. (2019)

The manual category in this taxonomy represents methods which require the learners' direct involvement in the process of engagement detection. It combines the methods self reporting and third-party observations, which were already emphasized as using different methods of annotation by Bosch (2016).

Semi-automatic methods, on the other hand, only require learners' indirect involvement. The only method contained in this category is engagement tracing, a method that utilizes the patterns of timing and accuracy in learners' responses to learning tasks.

Finally, the category of automatic methods, that is most relevant with respect to the previously introduced notion of Ongoing Assessment, includes the methods Computer vision-based methods, sensor-data analysis and log-file analysis. These methods do not require the learners' involvement as they extract features unobtrusively from automatically produced data. Computer vision-based methods apply computer vision techniques to video footage in order to estimate LE based on cues derived from gesture and posture, eye movement and facial expression. Methods using sensor data analysis use data from specialized physiological and neurological sensors. Finally, the log-file analysis methods evaluate learners' activity data preserved in log files. This type of data is typically produced by virtual learning environments like Moodle.

2.2 Ongoing and Digital Formative Assessment

Computer-based educational systems are used in education to provide direction and to manage instructions given to students. For example, Learning Management Systems (LMS) are suites of software that provide course-delivery functions. They are used for administration, documentation, tracking and reporting of e-learning programs as well as classroom and online events (Romero & Ventura, 2020). One of the most popular LMS around the world is the open source LMS Moodle. Today, there are more than 150.000 registrations worldwide with more than 200 million users attending more than 25 million courses¹.

With these numbers in mind, it is not surprising that there has been an increasing interest in how new technologies can support not only learning, but also assessment. DiCerbo et al. (2016) discuss the notion of Ongoing Assessment as one way how new technologies could improve teaching and learning. The authors refer to the following vision:

Imagine an educational system in which [...] students would progress through their school years engaged in different learning contexts, all of which capture and measure growth in valuable cognitive and noncognitive skills. [...] This vision does not refer to simply administering assessments more frequently (e.g., each week, each day) but rather continually collecting data as students interact with digital environments. [...] The vision involves high-quality, ongoing, unobtrusive assessments [...] that can be aggregated to inform a student's evolving competency levels [...] and aggregated across students to inform higher level decisions (Shute et al., 2016).

DiCerbo et al. (2016) expect Ongoing Assessment to have the potential to save time spent administering tests, to increase both the long-term retention and transfer of learning and to eventually measure progress towards educational goals without using tests at all.

The authors acknowledge, nevertheless, some hurdles to this vision of Ongoing Assessment. First, common measurements from different environments have to create a sufficient quality of assessment. Second, learning progressions need to be modeled in a way that can be applied across different learning activities and contexts taking into account the different pathways that learners can take. The third hurdle refers to impediments to moving forward the idea of the flipped classroom, whereas the forth hurdle is about parents' and other stakeholders' reservations about privacy, security and ownership regarding students' data.

The notion of Ongoing Assessment is closely related to the idea that the data arising from Computer-Based Educational systems can improve learning by helping to transform the way of teaching. This idea is the core of the concept of Digital Formative Assessment (DFA). Looney (2019) defines Digital Formative Assessment as follows:

Digital formative assessment includes all features of the digital learning environment that support assessment of student progress and which provide information to be used as feedback to modify the teaching and learning activities in which students are engaged. Assessment becomes 'formative' when evidence of learning is actually used by teachers and learners to adapt next steps in the learning process (Looney, 2019).

¹ Numbers from <https://stats.moodle.org/>

The idea of modifying the teaching and learning activities based on information acquired from students and used as feedback had already been established in the times of the traditional classroom as Formative Assessment (Black & Wiliam, 2010). Looney (2019) simply applies the definition of Formative Assessment from the traditional classroom setting to the context of Computer-Based Education. According to her, DFA has the potential to enhance learning, for example by providing more opportunities for learners to design their own learning goals and strategies while being provided with real-time feedback (self-regulated learning).

Engagement is essential for the concept of Digital Formative Assessment: Data can only be used as evidence in support of particular claims if learners engage in activities, thus generating data. Hence it is of major importance for assessments in Computer-Based Education that each specific learning task is designed in a way that it engages the learners in a way that it will help elicit evidence of their understanding. Only given such elicitation, the challenge of the task can be developed to involve learners actively in developing that understanding in the direction planned (Black & Wiliam, 2018).

2.3 Learning Analytics research

With more and more data becoming available from Computer-Based Education, research fields have emerged in order to explore and exploit this 'goldmine of educational data' (Romero & Ventura, 2020). One example is the field of Learning Analytics (LA), defined as

The measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimising learning and the environments in which it occurs [...] The value of analytics and big data can be found in (1) their role in guiding reform activities in higher education, and (2) how they can assist educators in improving teaching and learning (Long & Siemens, 2011).

The data that LA uses can basically come from any educational setting where students' activities are recorded by a Learning Activity Sensor (LAS), i.e. a learning platform-specific component for capturing data generated by the learners (Rabelo et al., 2015). Gathered data are for example demographic data, administrative data, the interaction between instructors, students and the educational environment and student affectivity (Romero & Ventura, 2020).

On one hand, Learning Management Systems (LMS) like Moodle have become the standard data source in predictive applications in LA (Di Mitri et al., 2017). Data collection has become very affordable since most LMS automatically log their users' activity. The information retrieved on the basis of the learners' digital traces is highly authentic as it reflects real and uninterrupted user behavior (Greller & Drachsler, 2012). Furthermore, instead of having to conduct a representative study, researchers now have longitudinal data at hand that comprise the entire student population (Drachsler & Kalz, 2016).

On the other hand, Pardo and Kloos (2011) already early stated that a significant part of the interaction during a learning experience takes part outside of LMS. The authors found evidence that only about 30% of the URLs that students use in a learning environment point to the institutional LMS. Taking into account only data obtained from LMS, the authors conclude, may therefore give only a partial picture of learning and lead to biased results. However, the number of data sources used in LA studies tends to be limited (Samuelsen et al., 2019).

A major challenge is that data is stored in different formats and often with multiple levels of meaningful hierarchy (e.g. answer level, session level, student level, course level), which makes gathering and integrating raw data from different sources a nontrivial task on its own (Romero & Ventura, 2020). These difficulties in integrating multiple data sources are related to the more general challenge of interoperability (Samuelsen et al., 2019).

2.4 The challenge of interoperability

Interoperability has been defined as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged". The benefits of interoperability, among others, can be seen in facilitated aggregation of data from different sources, increased durability of historical data and greater adaptability of IT applications (Cooper, 2014).

Increased interoperability would also benefit the sharing of data across institutions. This sharing, though object to ethical and legal concerns, is expected to give insight about the effects of different learning and teaching designs, about the effect of different contexts and about the behavior of groups of learners rather than the individual (Chatti et al., 2016; Drachsler & Kalz, 2016). Currently there are not many public datasets available, nor is there a specific LA datasets repository comparable to datasets repositories available for machine learning. In order to form such a future repository, datasets would ideally conform to an educational data specification (Romero & Ventura, 2020).

Even though many useful specifications exist (Cooper, 2015; Griffiths & Hoel, 2016), the Experience API specification (xAPI) has established in recent years as a *de facto* standard for describing analytically useful learning events. Formally speaking, xAPI describes the packaging and transmission of learner actions between a variety of LAS and a Learning Record Store (LRS), a database built to validate and store learning activity data (Kevan & Ryan, 2016). Core strength and weakness of xAPI is that it provides structural interoperability by defining the syntax of so-called activity statements and how to store and retrieve them while at the same time enabling users to create their own vocabularies (Griffiths & Hoel, 2016; Vidal et al., 2018).

Yet, in the interoperability model of the European Commission (2017), the use of data specifications only forms the first out of four layers of interoperability. Next to legal interoperability (challenged by incoherent legislation) and organizational interoperability (referring to proper documentation and service orientation), semantic interoperability stresses the challenge of common meaning. As Cooper (2014) points out, establishing common meanings to the data is typically the most difficult challenge with achieving interoperability.

Hence, it is most desirable to explicitly refer to a controlled vocabulary in order to avoid misunderstandings. These vocabularies are referred to as recipes and shared among the members of a so-called 'Community of practice' (Griffiths & Hoel, 2016). Additionally, an ontology allows for validation of the statements produced by a LAS in terms of both the requirements of the structural specification and the controlled vocabularies. The ontology-based validation of activity statements additionally provide the opportunity of machine-readable semantics which, in turn, allow for automated reasoning (Vidal et al., 2018).

If an adoption of an educational data specification is not possible, metadata information and a proper documentation increase the semantic and organizational interoperability of the data.

3 Related Work

3.1 Acquisition of log-file data from Moodle

This section covers the related work regarding the collection and processing of log-file data from Moodle. It presents different publications detailing how to carry out Learning Analytics with data from Moodle log files and examines how exactly this data is accessed, preprocessed, stored and consumed. It starts with a rather manual process and continues with more automatized approaches based on Moodle core functionality as well as solutions relying on extensions and external applications. This list of approaches to the Moodle log files covers the Moodle Analytics API, the Custom SQL queries report plugin and the Logstore xAPI plugin as the maybe most popular solutions. The section closes with a summary of the presented tools and publications.

3.1.1 Database query

In an early exploratory study, Romero et al. (2008) explain the necessary steps for data mining based on Moodle. More specifically, the authors point out the general importance of data preprocessing¹ which covers the tasks of user, session and transaction identification as well as data cleaning, reduction, transformation, enrichment and integration. Data preprocessing, the authors state, is normally a manual task carried out by an administrator. In an example the authors provide, they create an additional "summarization" table in the Moodle database where they store information from several other tables of the Moodle database at a student level. After discretizing continuous variables, they export this table together with Moodle's standard log table in ARFF format in order to analyze these data with the data mining tools Weka and Keel.

3.1.2 Moodle core functionality

The study of Akçapınar and Bayazit (2019) presents an easy way of obtain log-file data from Moodle: Log-file records can be downloaded as a file from Moodle by an administrator or instructor in CSV, JSON, ODS, XLSX, HTML or PDF format. In order to use their application MoodleMiner, log records have to be downloaded as a CSV file and then be uploaded to the application, where they are preprocessed automatically. The preprocessing tasks taking place in this step can be summarized as session identification based on login activity, manually entering grade information, splitting students into high- and low-performers and data aggregation on a student level. Nine features are extracted that way (it appears that the analysis is limited to these nine features). The resulting data are forwarded to the data mining part of the application which uses the additional software R as a data processing backend.

¹ A more comprehensive discussion of preprocessing specifically for Moodle can be found in Romero et al. (2014)

Another way to collect log-file data using Moodle core exists within the Analytics API² which became part of the Moodle core in version 3.4.0. Monllaó Olivé et al. (2018) introduce the underlying Analytics framework and present a use case (Identifying students at risk of abandoning a course). The so-called Analyser is the component that creates dataset files which contain targets and features supposed to be consumed by additional machine learning backends. It therefore defines the subjects of the data model and filters out the relevant data according to the configured targets, features and time range. It also prepares data for machine learning by applying data transformation techniques like normalization or one-hot encoding in case of discrete feature variables. Moodle core comes with a set of more than 50 features already, nevertheless, additional features can be programmed in PHP code.

3.1.3 Moodle plugins and blocks

A very flexible way to obtain log-file data inside the Moodle LMS is provided by the Custom SQL queries report plugin³ which allows to set up user-specific database queries which additionally can be scheduled to run, for example, once per week. Users can view the results of scheduled queries and trigger on-demand queries, while administrators can create queries and control who has access to them. In the case of scheduled queries, it is possible to add one row with every run to an existing table to keep track of changes in the results.

The plugin presented by Dondorf et al. (2019) is also implemented as a report plugin querying the Moodle database. It enhances the before mentioned report plugin by additionally logging technical details of the user sessions, an increased accessibility of the available reports for instructors and an advanced visualization of the results. The plugin provides predefined reports, but additional reports can be implemented in PHP code. The predefined report visualizes how values of the available indicators change as a course progresses over time.

The MDM tool (Luna et al., 2017) is a solution entirely implemented as a block for Moodle. Since Moodle courses are based on blocks that can be added, rearranged and removed, the MDM tool block can be added to any Moodle course by an instructor in the so-called Edit mode. Within the Data selection component of the application, data are collected from the Moodle database according to the subjects of the data model (students or resources), the time period and the set of predefined statistics the user chooses (In the case of student: Summary, Grades, Log, Forums, Quizzes and Files). These statistics can then be saved to an Excel file. This Excel file can be uploaded in order to carry out preprocessing steps, which are the following: Editing (e.g. adding grades), anonymization, discretization and splitting into training and test data. The resulting data are then saved to one or two Excel files before they are uploaded to the data mining component of the MDM tool.

Finally, there is the Logstore xAPI plugin⁴ that can be used to make Moodle emit log-file records as xAPI statements to an external Learning Record Store (LRS). It was developed by HT2Labs, the company that also provides the open-source LRS Learning Locker. Figure 3.1 shows the process of statement formation and emission as described by HT2Labs.

² https://docs.moodle.org/dev/Analytics_API

³ https://docs.moodle.org/38/en/Custom_SQL_queries_report

⁴ https://moodle.org/plugins/logstore_xapi

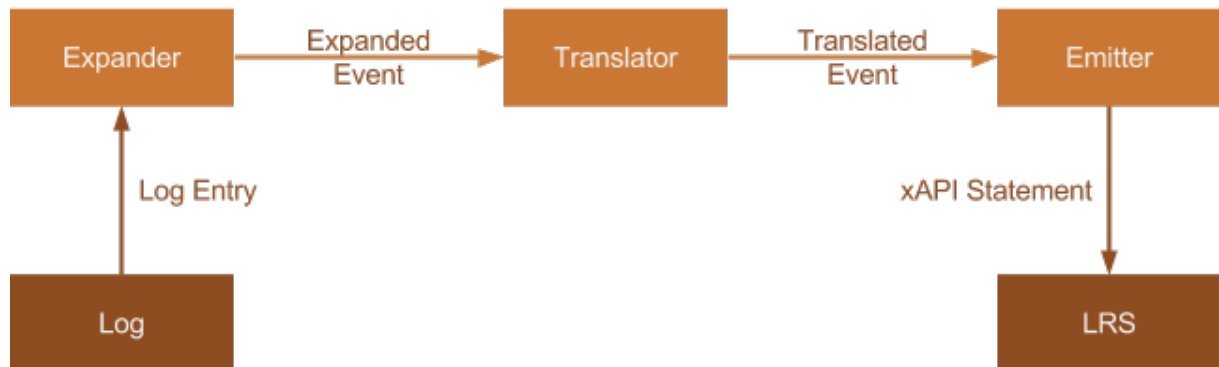


Figure 3.1: Process of xAPI statement formation and emission

Source: adopted from HT2 Labs, 2020.

According to the initial release note from 2015 (HT2 Labs, 2020), the plugin first passes a log entry from the standard log table in Moodle (`logstore_standard_log`) to the Expander, where it is enriched with additional data from the Moodle database. The expanded event is then passed on to the Translator which translates the expanded event according to the configured xAPI recipe. Finally, the translated event is passed on to the Emitter that forms the actual xAPI statement and emits it to the LRS.

The plugin's guide on how to deal with historical events⁵ reveals that the source of events has changed to a log table specifically destined for the Logstore xAPI plugin. It is explained there that the plugin receives events from Moodle and stores them in the database in a table called `logstore_xapi_log` which is later consumed by Moodle Cron. Therefore, the guide recommends to copy all historical events from the `logstore_standard_log` table to the table `logstore_xapi_log` in order to have them consumed too.

It should be mentioned that the number of supported events is still limited. Currently there is support for 65 events out of almost 500 events⁶ that Moodle provides through its Event API. It is possible though to add support for an additional event by adding an additional PHP file containing a so-called transformer function and mapping the event to this function. This way, every Moodle event can potentially be translated and emitted with the plugin.

3.1.4 External applications

The LeMo application (Fortenbacher et al., 2013) allows mining of log-file data coming from three different learning platforms, Moodle being one of them. In order to collect user data, the Moodle database is accessed directly and data are transferred to a relational database after being transformed into a unified data model. This data model stores log-file records as tuples of user, learning object and timestamp. Additional to these tuples, an action and the platform of origin are stored. An automatic preprocessing functionality is applied in order to exclude activity data not being caused by learners. The preprocessed data are then analyzed based on

⁵ https://github.com/xAPI-vle/moodle-logstore_xapi/blob/master/docs/historical-events.md

⁶ Existing events can be listed inside Moodle under Site administration > Reports > Events list

requests coming from the LeMo application, which also provides visualizations of the results.

Drăgulescu et al. (2016) present the CVLA tool, a solution that is operated through a frontend embedded in a Moodle report module. This application is able to include historical datasets from earlier Moodle versions of the same institution, treating the different Moodle instances as different data sources. For data mining applications that require data from various years, data from earlier Moodle releases are converted to a RDF data model using OpenRDF Sesame (today RDF4J), while the use of the D2RQ platform allows RDF-based real-time access to the database of the Moodle instance that the institution currently uses. In both cases, the authors employ an educational ontology to convert the data. Data are processed on a separate RDF storage server and the results are made available as a REST service in JSON format to the Moodle report module, including using Moodle core functionality for user identification and authorization.

A different approach is reported by Chaffai et al. (2017) who present a scalable infrastructure that can deliver real-time results. The application uses Apache Scoop and Apache Flume to move bulk and stream data, respectively, from the Moodle database to a column-oriented database. Table 3.1 shows the tables in the Moodle database that are used by the application, and which tables are mentioned explicitly by the authors as relevant for stream processing.

Table name	Description	Stream
course	Contains the list of courses	
user	Contains users information	
user_lastaccess	Contains the information about the last access	✓
context	Contains all context in moodle	
logstore_standard_log	Contains actions taken by users	✓
course_modules	Contains the list of course modules	
user_enrolments	Contains information about user registrations	
ressources	Contains the list of resources in the different courses	
role_assignments	Contains Information about the roles assigned to users	
role	Contains the list of roles that can be assigned to users	

Table 3.1: Tables for data collection from Moodle database

Source: adapted from Chaffai et al., 2017

Stream data coming in from Moodle's database are organized into topics using Apache Kafka. The data are processed by Spark Streaming, an extension of Apache Spark. Finally, results are stored to Apache HBase. The incoming results are immediately visualized by a live dashboard that reads from HBase using HBase thrift.

3.1.5 Summary

To sum up, many attempts have been made already to develop tools that make use of Moodle's log-file records. Existing approaches to acquire log-file data from Moodle heavily use direct access to Moodle's database, but also found various ways and formats for further processing, be it by Moodle itself, one of its extensions, an external application or the Logstore xAPI plugin.

The majority of the presented tools seem to work on a request basis, only the Custom SQL queries report plugin, the Logstore xAPI plugin and the application described by Chaffai et al. (2017) were explicitly build for fetching data periodically.

Some of the tools seem to ignore the opportunity to enrich Moodle's log data with context information (Akçapınar & Bayazit, 2019; Luna et al., 2017), which might be caused by the intention of their developers to create tools for instructors for which this context is obvious.

Likewise, two tools do not offer the opportunity to change the configuration in order to, for example, compute the values of a new indicator or feature. That might be due to the fact that these tools were explicitly developed for instructors (Dondorf et al., 2019; Luna et al., 2017).

The Logstore xAPI plugin is the only tool that is reported to make use of an educational data specification. Even though Fortenbacher et al. (2013) report using a unified data model that is somewhat similar to xAPI, their mapping of the different data sources into that data model remains unclear. Drăgulescu et al. (2016) however report that they transform data from different Moodle instances to RDF making use of an ontology and that data are made available as a REST service in JSON format. Their tool can therefore be considered to produce an interoperable data output as well.

Finally, it should be mentioned that only a limited number of tools and applications could be reviewed in this section. An overview about additional tools that are useful for the acquisition of Moodle log data can be found in Slater et al. (2017). An investigation of the strengths and weaknesses of a smaller number of tools is presented in Conde et al. (2015).

3.2 Learning Engagement detection using log-file data

This section reports about existing approaches to detect engagement based on Moodle log records. Several studies report about applying log-file analysis in order to detect Learning Engagement or related constructs. Aluja-Banet et al. (2019) present an "approach on how the usage of VLEs (Virtual Learning Environments) can be traced and used to enhance the teaching quality so as to improve engagement". Feild et al. (2018) design, train and test a generalized disengagement classifier using a large-scale dataset in order to identify student disengagement within the learning tool. Ferguson and Clow (2015) try to replicate the results of a disengagement classifier developed for one MOOC (massive open online course) platform using data from another type of MOOC platform. Finally, Wise et al. (2014) describe their attempts to "make log-file trace data of speaking and listening activity visible to learners".

Aluja-Banet et al. (2019)

To my knowledge, Aluja-Banet et al. (2019) are the only authors reporting the measurement of a Learning Engagement related construct based on log-file data obtained explicitly from a Moodle LMS. The authors state that they aim at "a tool to measure in (quasi) real time the motivational state of students from the digital traces left by them in the VLE [...] for every student, in every subject" in order to "look beyond the traces" and provide teachers and stakeholders with objective information about students' disengagement.

The authors base their research on motivational theory that decomposes motivation into three components:

- Activity
- Persistence
- Intensity

Activity refers to the initiation of a productive behavior, while persistence focuses the maintenance of this behaviour and the effort to overcome obstacles that may arise. Intensity relates to the rather cognitive and emotional investments in the pursuit of a goal.

Based on these three components of motivation, the authors present four groups of behavioral indicators:

- Speed indicators
- Persistence indicators
- Intensity indicators
- Choice indicators

Speed indicators measure "how fast an individual starts the task after its recognition, how fast an individual completes the task, and how fast the student moves from one task to the next one." They refer to the 'activity' component, following the idea that quick action is an indicator of motivation.

Persistence indicators, on the other hand, measure the degree of determination of a student to take effort. They correspond to the 'persistence' component and serve as a key indicator of motivation in taking up and overcoming challenges.

Intensity indicators measure the intensity of the taken effort. They are measures for the 'intensity' component and follow the idea that the effort taken in order to achieve a certain goal can be interpreted as an effect of the motivational level of the agent.

Choice indicators, finally, were added as a fourth group of indicators. They do not correspond to one of the before mentioned components of motivation, but were added to also take into account the fact that learning efforts reflect the agents' choices out of a variety of options. They predominantly measure non-expected but desired performance.

Table 3.2 details the behavioral indicators that the authors report for each of the four groups. Aluja-Banet et al. (2019) use these indicators to compute an index. In the discussion of their results, they call the derived "composite index of motivation" the main output of the system. They present the psychometric parameters of the measurement of the motivation construct, which imply that only six of the indicators were included in the composite index. Those indicators are:

1. Agility rate
2. Resilience level
3. Engagement level
4. Competitive level
5. Curiosity rate
6. Forum access

Indicators group	Indicator
Speed indicators	Agility Rate, measured as a sigmoid function of the difference between the date on which a task became available in respect of the first access of that task, by a student.
	Time spent, measured as the total time spent on the VLE to complete a task.
	Transition time, measured as the time spent between consecutive tasks on the VLE.
Persistence indicators	Number of logs executed in a given task per day.
	Resilience level, computed as the ratio of the maximum number of tasks performed in the interval of two hours in respect to the total number of tasks performed in a day, in a given subject. This indicator reflects persistence in doing the tasks.
	Number of attempts used to submit a given task.
	Persistence level, measured as the mean time used between consecutive attempts at a given task.
Intensity indicators	Delivery rate, computed as the ratio of the fulfilled obligatory tasks in respect to the pending ones in a given subject. This indicator reflects required performance.
	Engagement level, computed as the ratio of all activities performed in one day for a given subject in respect to the maximum activity in that subject the past two weeks. This indicator reflects peak performance.
	Competitive level, computed as the ratio of the total of the activities performed by a student in a given subject, in respect to the most active student on the same day. This indicator reflects comparative performance.
Choice indicators	Curiosity rate, computed as the ratio of the fulfilled nonmandatory tasks in respect to the pending ones in a given subject. This indicator reflects expected, but not required performance.
	Forum access. It measures the instances of accessing a Forum related to activities of a given subject on particular day. It reflects non-expected but desired performance, although passive.
	Forum participation. It measures the participation in a Forum related to activities of a given subject on particular day. It reflects active non-expected but desired performance.
	Priority rate, computed as the ratio of log access to any activity, on a given subject, in respect to all the logged access to that subject, in the particular day. It reflects the student's prioritization of a given subject.

Table 3.2: Behavioral indicators of motivation by group*Source:* Aluja-Banet et al. (2019)

Aluja-Banet et al. (2019) indicate that the final composite index is "easy to understand and very informative", even though they acknowledge that teachers have to operate on the same level of Moodle usage so that their performance can be compared. Unfortunately, the authors do not report whether indicators are weighted before they are aggregated into the index.

The authors emphasize that the indicators' values need to be normalized to compare them to each other. Furthermore, they state that the indicators need to be measured at least daily for each student and per course in order to inform about the evolution of motivation in each course and to allow interventions. However, three of the indicators (resilience level, engagement level and priority rate) are reported to be calculated using another period of measurement or taking into account all courses at once.

Feild et al. (2018)

Feild et al. (2018) use data from the Connect assignment and assessment platform by McGraw-Hill Education. They develop a generalized disengagement classifier in order to identify student disengagement at scale within the learning tool. To train and test the classifier, they can use a very large dataset containing data of more than 4.5 million students.

The authors report to have hand selected the features according to whether the distribution of values was significantly different in the dataset between disengaged and engaged students. A disengaged student was defined in this study as

A student who submits assignments through the online learning tool up until a time t and never submits another assignment has disengaged at point t .

The authors report that ten features could be derived, each of which improved significantly the performance of the classifier. These ten features are

1. Average grade
2. Average time spent on assignments relative to class average
3. Percent assignments submitted
4. Percent assignments submitted late
5. Percent assignments submitted on time
6. Percent assignments not submitted
7. Percent assignments submitted late or not at all
8. Days since last submission
9. Submission time relative to due date
10. Maximum number of consecutive late assignments

The first feature suggests that not only log-file data were used in order to obtain features relevant for the classifier. In fact, it makes sense that past grades appear to be related to disengagement. Nevertheless, the nine other features that Feild et al. (2018) report to have used for their classifier are most likely available from log files.

Ferguson and Clow (2015)

In their replication study, Ferguson and Clow (2015) use log data obtained from four FutureLearn MOOC platforms in order to replicate the results of a disengagement classifier developed for the Coursera MOOC platform. MOOC is an abbreviation for 'massive open online course', which specifies these platforms to deliver their learning content

- at scale (to many students at once)
- without formal entrance criteria
- through the internet and
- structured as courses.

In the Coursera study, students' activity had been assigned each week one of four categories:

- T = on track
- B = behind
- A = auditing and
- O = out.

The obtained category sequences (for example, [O,T,T,T,T,B,A] for an eight-week course) are used to partition into groups using a k-means clustering algorithm. However, the authors report that this approach can only reproduce two out of the expected four clusters (because I am focusing here on how engagement was measured, I will not detail the clusters themselves).

According to the authors, this might be related to an important difference between FutureLearn and Coursera: Coursera course content focuses on videos and implements a rather instructivist approach of teaching. FutureLearn courses, on the other hand, make substantial use of other learning materials, predominantly text with pictures. In addition, the FutureLearn employs a social-constructivist pedagogy, aiming at a considerably higher amount of conversations as well as the provision of tools for commenting, responding and reflecting of course content.

In response to the mixed results they obtain from direct application of the Coursera study's method to the FutureLearn data, the authors develop a new method that can better reflect the importance of discussions in MOOCs that follow a social-constructivist approach. The researchers calculate a weekly score for each student based on points assigned as follows:

For each content week, students were assigned the value 1 if they viewed content, 2 if they posted a comment, 4 if they submitted the week's assessment late and 8 if they completed the final assessment before the end of the week in which it was set (i.e. early, or on time). These values were added up to give a total for each week.

Ferguson and Clow (2015) explicitly state that there is no way to tell from the log data whether learners have participated in the discussion by reading but not by adding a comment. The resulting possible weekly scores are shown in Table 3.3.

Score	Interpretation
1	only visited content (for example, video, audio, text)
2	commented but visited no new content
3	visited content and commented
4	did the assessment late and did nothing else that week
5	visited content and did the assessment late
6	did the assessment late, commented, but visited no new content
7	visited content, commented, late assessment
8	assessment early or on time, but nothing else that week
9	visited content and completed assessment early / on time
10	assessment early or on time, commented, but visited no new content
11	visited, posted, completed assessment early / on time

Table 3.3: Possible weekly engagement scores

Source: Ferguson and Clow (2015)

Feeding the derived sequences to the k-means algorithm again, Ferguson and Clow (2015) obtain seven clusters, which they are able to construe as differently engaged groups. According to the authors, the distinct patterns observed for the Coursera and the FutureLearn MOOC platforms suggest that patterns of engagement are influenced by decisions about pedagogy.

Wise et al. (2014)

It was already mentioned in the subsection on Ferguson and Clow (2015) that the authors acknowledged that there is no capture in the log data of whether students participated in the discussion by reading but not by adding a comment. Wise et al. (2014) emphasize that both contributing and attending to discussion messages is an important activity in realizing the theoretical potential of online discussions to support knowledge construction. The authors hypothesize that the bulk of research focusing on posting activity may be related to the fact that 'speaking' in online discussions is clearly visible, while 'listening' remains largely invisible.

Wise et al. (2014) use data obtained from the Starburst discussion forum, software originally developed to present discussion threads as tree structure that allows students to see the structure of the discussion and their own comments' location within it. In their study, they examine the effects of presenting the results of 'embedded' and 'extracted analytics' to students. I will only focus on the 'extracted analytics' here.

The metrics for the 'extracted analytics' were chosen because they had been among the variables in the E-Listening research project that provide "a useful signal with a minimum of noise" and are additionally readily understandable. The final metrics presented in Table 3.4 are supposed to "make log-file trace data of speaking and listening activity visible to learners".

Metric	Definition
Range of participation	Span of days a student logged in to the discussion.
Number of sessions	Number of times a student logged in to the discussion.
Percent of sessions with posts	Number of sessions in which a student made a post, divided by his/her total number of sessions.
Average session length	Total length of time spent in the discussion divided by his/her number of sessions.
Number of posts made	Total number of posts a student contributed to the discussion.
Average post length	Total number of words posted by a student divided by the number of posts he/she made (including manually inserted direct quotes).
Percent of posts read	Number of unique posts that a student read divided by the total number of posts made by others to the discussion.
Number of reviews of own posts	Number of times a student reread posts that he/she had made previously.
Number of reviews of others' posts	Number of times a student reread others' posts that he/she had viewed previously.

Table 3.4: Discussion participation metrics

Source: adapted from Wise et al. (2014)

The authors report that they obtain these metrics by applying a combination of MySQL database queries and VBA macros. As a result, nine values are created for each student (one for each metric in Table 3.4). The metrics are reported to each learner in a summary table. This table does not only contain the student's individual metrics, but also the course averages as a local reference for the student.

3.3 Contribution of this project

Considering the above mentioned research, the contribution of this thesis is twofold:

First, it reports the development of a flexible assessment solution for Moodle that can perform the assessment of an arbitrary number of constructs. This solution offers scheduled construct assessments and thereby implements the notion of Ongoing Assessment (see section 2.2).

Second, it exemplifies the developed assessment solution for Moodle by the configuration of scheduled assessments of the Learning Engagement construct. It thereby demonstrates how interoperable assessment results for virtually any construct can be obtained periodically for purposes of Digital Formative Assessment and Learning Analytics (see sections 2.2 and 2.3).

4 Concept and Design

In this chapter I discuss the concept and design of the project that I implemented. After introducing the epics that represent the main intentions behind this project from the perspective of my supervisor, I present the main features that the project was designed for. I then continue with the list the acceptance criteria for these features. The fourth section gives an overview about the considerations that influenced my decisions for design, from which I present the major ones.

4.1 Epics

In order to get an overview about the requirements of the project, the global features of the application were derived from discussions with my supervisor about its intended purpose. From these discussions, I could identify three main intentions of my supervisor which I present here as epics:

- Assessment scheduling mechanism
- Assessment performance, storage and reporting
- Provision of results

These three epics represent the core functionality of the project. The entire workflow is intended to be triggered by a schedule configuration, which is expressed by the first point 'Assessment scheduling mechanism'. The second point, 'Assessment performance, storage and reporting' refers to the intention to obtain data from a Moodle LMS for the calculation of indicators, to the aggregation of these indicators into constructs, to the storage of these aggregated results and to the reporting about the process. Finally, the last point 'Provision of results' refers to the intention that the results of the assessment are exported to and persisted in an external database.

4.2 Features

I used these epics to group the features the project needs to have. Each of the features will be detailed briefly in the following subsections.

4.2.1 Pending assessments identification

The assessment workflow needs to take into account for which courses and users in these courses an assessment should be carried out. Therefore, out of the courses and users in the connected Moodle LMS, a configuration needs to be provided where courses and, in these courses, users can be selected for assessment. Secondly, the assessment procedure needs to make use of this configuration in order to deliver results only for the courses and users selected.¹

4.2.2 Assessment scheduling

In order to allow for an automated assessment, a functionality is needed where the user can plan assessments to take place repeatedly at a certain time. If possible, the configuration should also allow to schedule the assessment of different constructs at different times. The scheduled assessments should take place completely automatically so no further user input is necessary in order to complete the assessment. Additionally to the scheduling configuration, the project should provide the option to turn the scheduling function on and off separately for each construct.

4.2.3 Assessment performance

In order to deliver assessment results for learners based on their activity, the project needs to calculate indicator values for each indicator associated with the assessed construct, and to aggregate them into an assessment result for each user in each course that was configured accordingly (4.2.1). All assessment results are to be returned together (for example, as a dictionary) so they can be showed in the dashboard (to "stage" the results of the current construct settings) or be stored to the database in a single pass.

4.2.4 Results storage

The assessment results are to be stored to the database so they can be reported or exported independently from the assessment process at a later stage. This will additionally allow to look up, for example, whether a learner was assessed and what his value is right after an assessment has taken place, and maybe also to provide these data to an external system. If the assessment for courses and users has been disabled (4.2.1), the assessment results already existing in the database for these courses and users should ideally be deleted.

¹ Initially, this feature was about the identification of whether learners had shown activity in the system relevant to one or more indicators of a construct, in order to determine the necessity of a reassessment. This was changed after feedback suggested that assessment should always take place at a course level but provide an option to enable/disable the assessment for users and/or complete courses.

4.2.5 Results report

The results of the assessment are to be reported in order to see when the last assessment took place and what the respective values for the learners are. An option to start an ad-hoc assessment without saving to the database is very desirable for indicators as well as constructs because it would allow to immediately see what results the indicator or construct will deliver in the future (4.2.3). The report should provide the course, the user and a timestamp together with each result.

4.2.6 Results emission

The assessment results are to be exported to and persisted in an external database, where they will be available for further analysis, for example together with xAPI statements coming from the Moodle LMS and data coming from other Learning Activity Sensors. Under no circumstances may assessment results be lost in the process of the export. It should be made sure that assessment results will be resent at a later time if the external database is not available. In general, the availability of the external database should not affect the assessment performance.

4.3 Acceptance criteria

In order to verify whether a feature was successfully implemented, I developed acceptance criteria for each feature. That way, an objective measure is provided that can be used to check whether the project has the intended features.

Table 4.1 shows for each of the features from section 4.2 the acceptance criteria. These acceptance serve as criteria in section 6.1 as proof of whether the features were implemented appropriately.

Epic	Feature	Acceptance criteria
Assessment mechanism	Pending assessments identification	The project provides the option to set whether or not a course, and in this course, a user will be object to future assessments. Scheduled assessments are performed automatically only for those courses and users that were previously selected.
	Assessment scheduling	Construct assessments are performed automatically following a configurable schedule without further user input being necessary. The scheduling of the construct assessments can be turned on and off.
	Assessment performance, storage and reporting	Assessments can be started by calling a method and the aggregated values for the construct / original values for the indicator are returned according to the pending assessments identification. All assessment results are returned together in one single data structure.
Provision of results	Results storage	Results of construct assessments and indicator calculations can be viewed when connecting to the database. Ideally, the assessment results already existing for a course or user for whom assessment was disabled are deleted from the database.
	Results report	Course- and user-specific assessment results including a timestamp of the assessment are reported after an assessment has taken place. Ideally, an ad-hoc assessment can be started and the results be viewed without saving to the database.
	Results emission	Assessment results are exported to and persisted in an external database. If the external database is unavailable, this does not affect the assessment performance. In that case, scheduled assessments continue to take place. Assessment results are not exported only until the external database becomes available again.

Table 4.1: Features from section 4.2 and their acceptance criteria

Source: own conception

4.4 Design objectives and decisions

4.4.1 Flexibility

It was aimed at a high flexibility considering the original purpose of the project: To allow for an iterative development of the assessment approach. Therefore, the configuration of constructs was designed with the objective of a dynamic evolution of constructs and their measures. This finally led to the decision to allow for the creation of constructs and indicators at runtime, including the necessity to execute code that was entered for the indicators by the user.

Additionally, it turned out during development that the process of transforming the indicator values' scale before aggregating them into construct results would also benefit from a higher degree of flexibility. The design was therefore adapted in order to provide the user with options of what exact transformation should be applied to the indicator values.

Finally, it was also an objective to allow for flexible scheduling of construct assessments. Resulting from the decision that constructs and indicators could be created dynamically, a separate schedule would need to be created for each construct and indicator in order to provide the flexibility to configure the scheduling of each construct and indicator individually.

4.4.2 Extensibility

Some of the functionality aimed for in the software design could not be realized in this project due to time constraints. Nevertheless, it was tried to design for this functionality anyway so it could be realized more easily at a later stage.

One example for this is that, as already mentioned in 4.4.1, during development it was discovered that the data transformation process may be insufficient and benefit from more powerful data transformation tools. As it was not sure at this time whether the made adjustments would be sufficient, it was decided to pave the ground for any additional transformations using numpy and scikit-learn methods by transforming the values to a pandas data frame.

Another example is that it was planned originally to provide the option to the user to schedule separate construct assessments for each course (in order to provide even greater flexibility than described in 4.4.1). When the feedback after the interim presentation suggested that this functionality was not a high priority, it was decided to not to remove the 'courses' parameter from the methods that are used to calculate and store the results for the construct assessments. This parameter could be used to narrow down the results that these methods return in the future.

4.4.3 Reusability

Another design objective was to reduce the redundant efforts in the configuration of the constructs' measures. Once indicators have been configured, the project should allow to reuse these indicators instead of having to reconfigure them every time a similar indicator is needed. Construct-indicator relationships were therefore designed as many-to-many relationships. Also, the weight of a construct's measuring indicators was determined to be an attribute of the construct-indicator relationship.

Reusability was also pursued in the source code. For example, the execution of the code that was entered for the indicators by the user in order to deliver ad-hoc indicator results was originally designed as a method of the `IndicatorResult` class. The fact that this function was redesigned as a `calculate_result()` method of the `Indicator` class that additionally accepts two parameters `minutes` and `courses` facilitated the reuse of this method in another scenario: It could now additionally serve to provide the indicator results for construct assessments.

4.4.4 Modularity

For the sake of manageability in terms of implementation and refactoring, it was decided to separate the project logically into different apps. This separation represents on the one hand the idea that an app serves a specific purpose within the project, on the other hand follows the principle that links between classes, functions and data inside an app are stronger than between apps. Beyond that, the logical separation is supposed to help increase orientation. As a result, the 'app' project was designed to consist of the following apps:

- `data_source`
- `administration`
- `assessment`
- `export`

Among the apps, 'data_source' is the one that provides access to Moodle's database. The 'administration' app is the place where Moodle's courses and users are imported to the application, where the enabling of the assessment for these is provided and from where this configuration can be requested. The configuration of indicators and constructs, the generation of results, as well as their storage and reporting take place inside the app 'assessment'. Finally, the 'export' app exports the historical results to the external database, cleans up after their export and provides the results of the latest construct assessments through a REST API.

Additional modularity is given by the files and folders inside the Django apps:

- `admin.py`
- `apps.py`
- `db.py`
- `forms.py`
- `models.py`
- `serializers.py`
- `tasks.py`
- `tests.py`
- `urls.py`
- `views.py`
- `migrations/`
- `templates/`

The app-specific models (corresponding to tables in the database) are defined in `models.py` and typically correspond to the folder `migrations/`, which contains the migration files used to configure the databases. The app's models can be populated with data either by using the form classes in the app's `forms.py` file (typically in a view) or by registering them in the app's `admin.py` file, which makes them available in Django's `admin.site` module. The data from the models are typically presented to the user by a view class in the app's `views.py` file, rendering a view-specific template from the app's folder `templates/`. The file `urls.py` defines what URL is routed to which of the app's views. Finally, the file `apps.py` configures the app itself (in this project, most often only the name) and the file `tests.py` is where the app's unit tests are defined, which was not used in this implementation.

A `db.py` file contains additional database configurations, if needed. Similarly, a `serializers.py` file is added when there is need to make output available in JSON format.² If there are celery tasks that are to be made available for scheduling, they are defined in the `tasks.py` file.³

In the 'assessment' app, an additional logical separation was introduced for the files `models.py` and `views.py` in order to increase orientation as files grew too large. Since the classes were all either related to the `Indicator` or the `Construct` class, it was decided to create two folders `models/` and `views/` and to create files `indicators.py` and `constructs.py` in each of these two folders.

4.4.5 Compatibility

It was aimed from the beginning to allow for exchange of the default Moodle database and the export database for other databases. It was thus planned that the project would be delivered as a readily configured system, coming together with a Moodle and its database as well as a NoSQL database for the assessment result exports, but that the exchange of these databases should be facilitated by providing the connection parameters in the `.env` file of the application.

The question of how to provide the assessment results is very close to the topic of interoperability. The objective here was to allow for any applications to access these results. Hence it was decided to use a Cassandra database to export and persist the historical results of the assessments. During development, it was decided to additionally offer the last assessment results of each construct through a REST API in JSON format, which is an output format that is widely accepted and easily transformable and is therefore expected to be able to serve a variety of applications.

It was also considered whether the envisioned project should use Moodle's Data Manipulation API⁴ in order to abstract from the different types of possible Moodle databases (MySQL, PostgreSQL, Microsoft SQL Server, MariaDB, or Oracle). This would have increased the compatibility and interoperability of the implemented project as well. Because of time constraints, this approach could not be followed in this thesis. It was assumed instead that Moodle uses a MySQL database.⁵

² This is a convention of the Django REST framework, see 5.1.3

³ This is a celery-specific convention, see 5.1.5

⁴ https://docs.moodle.org/dev/Data_manipulation_API

⁵ The use of Django as basic framework and the decision to provide the access to the Moodle database in a separate 'data_source' app should nevertheless allow to switch the type of database with minimal code changes.

5 Implementation

In this chapter I provide a comprehensive look at the implementation process. It starts with an overview about the used technologies. The second section presents the different components of the project. The last section details the project workflow.

5.1 Technologies

5.1.1 Docker

The project was designed as microservice, planned to work as an independent part of a bigger system that collects the generated assessment results. In order to allow for easy deployment, it is provided as container in a Docker setup, accompanied by a Moodle instance based on Apache, a MySQL database serving the Moodle container and a Cassandra database as storage for the historical assessment results. Finally, a rabbitmq message broker serves for to sending and receiving task messages.

The Docker images of these services are defined in the file `docker-compose.yml` which is also used for startup of the project. This file determines where the different services create their own volumes in order to persist their data, which environment variables they use for configuration and which commands are to be run in the startup process in order to provide their service. It also contains commands that will make services wait for each other at startup in order to ensure that they are ready for incoming connections (especially relevant for the databases). If one of the services crashes, the `docker-compose.yml` file determines that it will be restarted automatically unless it was stopped by a Docker command.

The image of the project itself is called 'assessment_ms' and is built when the `docker-compose` file is executed the first time on Docker, whereas the images of the other services are downloaded from Docker hub. For the build of the `assessment_ms` image, a `Dockerfile` is provided in a folder with that same name. This `Dockerfile` will be described in the following subsection. The `docker-compose.yml` file and the `.env` file as well as the project's `Dockerfile` and the `requirements.txt` file of the `assessment_ms` image are included in the appendix to this thesis.

5.1.2 Python

The `Dockerfile` of the 'assessment_ms' image is built on top of an official Python Docker image tagged '3.7-slim', which is a lower-size version compared to the standard 3.7 Python image.¹

¹ It was first envisioned to use the even-smaller 3.7-alpine image, but incompatibilities with the pandas requirement enforced the use of the larger 'slim' version.

The Dockerfile first commands to upgrade `pip`, to disable all buffering of the python output, and to copy and run the installation of the requirements defined in a `requirements.txt` file located next to the Dockerfile (the contents of this file will be mentioned in the following subsections). After installation of the requirements, a directory `'app'` is created and the project source code (identified by env variable `ASSESSMENT_MS_SOURCE`) is copied to this directory. After building of the image, Docker starts the container and starts the web server. The project becomes available at port 8000 of the local machine as specified by the `docker-compose.yml` file.

5.1.3 Django

Django 2.2 serves as basic framework of the project.² Django is based on Python and therefore offers all Python functionality, which makes it very powerful. Nevertheless, Django is a powerful framework itself, with its object-relational mapping (ORM) being probably one of its most appreciated features. One other advantage of Django is that is quite popular and therefore embedded in a rich ecosystem of third-party extensions that the project makes use of.

A SQLite database was configured as `'default'` connection to store its data, but also a MySQL database (as `'moodle'` connection) (using the MySQL Connector/Python driver³) and a Cassandra database (as `'cassandra'` connection), both coming together with the project. Since there is no support for a Cassandra backend in Django core, the third-party package `django-cassandra-engine`⁴ was used that depends on the DataStax Python Driver⁵.

Additionally, several extensions were used to provide additional functionality. In order to provide configuration of central aspects of the Django project from the Docker `.env` file, `django-environ`⁶ was added to the project. This package is especially used to provide control over settings like Django's `DEBUG` parameter and the `connection` parameters for the external databases that can be changed in the `.env` file instead of having to look for them in the source code of the project.

The package `django_ace`⁷ was added in order to provide a code editor inside the Django web app that displays the Python code entered for each indicator with an appropriate formatting and to offer additional code-editor specific functionality like for example the replacement of tabs with spaces.

In order to provide web APIs, the Django REST framework package⁸ was included. This package makes it possible to serialize contents from the database, to handle requests and to provide the serialized contents in a view. This was used in the implemented project to provide the latest assessment results of each construct in a REST API.

Finally, the `django-celery-beat` extension⁹ was added in order to use Django's ORM functionality as a backend for the scheduling of periodic tasks in Celery (see 5.1.5).

² Even though with Django 3.x a newer major version was available since the end of 2019, Django 2.2 was chosen because not all third-party libraries already offered Django 3.x support.

³ <https://github.com/mysql/mysql-connector-python>

⁴ <https://github.com/r4fek/django-cassandra-engine>

⁵ <https://github.com/datastax/python-driver>

⁶ <https://github.com/joke2k/django-environ>

⁷ <https://github.com/tarekbecker/django-ace>

⁸ <https://github.com/encode/django-rest-framework>

⁹ <https://github.com/celery/django-celery-beat>

5.1.4 Scikit-learn

After the presentation of intermediate results of the development process, feedback was obtained suggesting that the implementation of the aggregation of indicator results into construct results at this stage (using the weighted mean of all indicator results associated with the construct after normalizing them in the context of the course) may produce biased construct results.

This scepticism was mainly induced by the existence of different scales of the indicators and the prevalence of outliers in their results. The critical process in the aggregation of the results was therefore referred to as 'preprocessing'. It was thus decided to make use of a preprocessing library, leaving the choice of which exact preprocessing method to apply to the user. The python module scikit-learn¹⁰ seemed to offer enough flexibility to allow the user to find the adequate preprocessing method¹¹ of his choice and to be powerful enough in case that a further increase of the calculation complexity would become necessary.

To be able to use the preprocessing library, the package Numpy¹² was added (scikit-learn depends on it anyway). Pandas¹³ was included to use the option to operate on data frames when applying the preprocessing. This also provided powerful options for data handling in case of any further improvements. Since it was recommended for pandas, the packages six¹⁴, pytz¹⁵ and python-dateutil¹⁶ were included as requirements as well.

5.1.5 Celery

An important feature was that the project needs to schedule construct assessments. I therefore looked for a task queue that integrates with Python and Django. The ideal scenario would be that task scheduling could be configured together with indicators and constructs.

A solution was found in using celery¹⁷ together with the Django extension django-celery-beat¹⁸. Celery is Python based, which makes it possible to configure it inside the Django app, and the extension allows to store periodic tasks and their schedules in Django's database backend. Additionally, it offers several ways to schedule tasks via request, for example by using celery flower¹⁹ (which was not implemented but could be object to future work).

Celery uses a message broker to deliver task messages to dedicated worker processes. Since no messaging infrastructure existed yet, a rabbitmq container was added to the docker-compose setup to serve the project as a message broker. Potentially, the project could also use an existing messaging infrastructure, which would also allow other applications to put messages on the project's task queue.

¹⁰ <https://github.com/scikit-learn/scikit-learn>

¹¹ <https://scikit-learn.org/stable/modules/preprocessing.html> explains the different options. Additionally, https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html provides an illustration of the effect of the available preprocessing methods on data with outliers.

¹² <https://github.com/numpy/numpy>

¹³ <https://github.com/pandas-dev/pandas>

¹⁴ <https://pypi.org/project/six/>

¹⁵ <https://pypi.org/project/pytz/>

¹⁶ <https://pypi.org/project/python-dateutil/>

¹⁷ <https://github.com/celery/celery>

¹⁸ <https://github.com/celery/django-celery-beat>

¹⁹ <https://github.com/mher/flower>

5.2 Project and apps

5.2.1 'app' project

The Django project itself is the place of project-wide settings, definitions and configurations of higher-level services used by its apps.

The project contains the following files:

- settings.py
- db.py
- forms.py
- celery.py
- urls.py

The file settings.py contains project-wide settings. Most important of these is the definition of the project's `connections`. All three databases are configured as database backends for the project, so that they become available as `connections` and Django's ORM functionality can be used. The resulting `connections` are 'default', 'moodle' and 'cassandra'.

The file db.py defines the database routing, i.e. it tells Django which `connection` to use in what case. Since the modular structure of the project (see section 4.4.4) reflects the separation into an input app, two processing apps and an output app, it was possible to define database routing based on the app in which a model is defined: The 'data_source' app's models are routed to the 'moodle' connection, models defined in the 'export' app are routed to the 'cassandra' connection. All other models are routed to the 'default' connection.²⁰

The forms.py file only appears here in the project because Django forms do not provide placeholders. It defines a form class that allows using placeholders in HTML forms, making the input forms look slightly nicer. This form class is then used in the apps wherever data can be entered using a form, for example when creating indicators and constructs.

The file celery.py configures the celery service that provides the task scheduling functionality for all apps. Those apps have by convention their task functions in a file called tasks.py, decorated as `@shared_task` in order for them to be located by the celery service.

The file urls.py is the root URLconf file as defined in the settings.py file. Together with additional app-specific url routing files it includes, it defines which urls are provided and to which view classes they are routed to.

The templates folder, which is also situated on the top level, includes the basic templates that can be used or extended by all apps. Additionally, each app can have its own templates folder.

Finally, the project is accompanied by two more folders next to the already mentioned 'templates' folder: 'fixtures' and 'logs'. The 'fixtures' folder contains a file that can be used to initially populate the 'default' database with a construct, its indicators, periodic tasks and schedules. The 'logs' folder is where the log files are written to as defined in the settings.py file.

²⁰ The database routing can still be overwritten by calling the `using()` method in Django queries.

5.2.2 'data_source' app

Among the project's apps, 'data_source' is the one that provides the other apps access to Moodle's database. Among the app's files, the following two will be discussed here in detail:

- models_abstract.py
- models.py

A few number of model classes that are to be inherited from inside other apps of the project are defined as abstract models in the models_abstract.py file. 'Abstract models' means here that they only define a structure that can be inherited from, but not correspond to a database table themselves. The main purpose for this procedure is to create a single point of definition for these models as Django does not provide a mechanism for the dynamic inspection of an external database.²¹ Instead, models need to be defined explicitly, making it necessary to change the apps' models in case of changes in the structure of the external database.

In order to be able to dynamically obtain all the classes that are defined as abstract model classes in the models_abstract.py file, these classes all inherit themselves from an abstract class called `AbstractMoodle`. This is used when actualizing data inside the 'administration' app.

The app's own models are to be found in the models.py file. As far as they were defined in the models_abstract.py file, they inherit from the abstract model classes there and just add the `_meta` attribute 'db_table', which defines the table name of the corresponding table in the database corresponding to the 'moodle' connection. Additionally needed models can be defined here directly (without inheriting from the model classes in the models_abstract.py file).

5.2.3 'administration' app

The app 'administration' is the place where Moodle's courses and users are imported to the project, where the enabling of the assessment for these is provided and from where this configuration can be requested. Among the app's files, the following three will be discussed here in detail:

- models.py
- tasks.py
- views.py

The models.py file defines the models necessary for the import and configuration of Moodle courses and their users into the project. As already mentioned in 5.2.2, this is achieved by inheritance from the abstract model classes defined in the 'data_source' app's models_abstract.py file. More specifically, the following classes are inherited from the 'data_source' app's class `AbstractMoodle`:

²¹ There is an 'inspectdb' mechanism in Django, but as it is a management command, it can not be used to obtain the structure of the tables of the external database at runtime. Instead, it can be used to obtain a first draft of the model specifications as they might be needed in Django whenever to 'moodle' database is changed to a non-MySQL database. To create this code in a file called models_new.py, run the following command on the terminal: `docker exec assessment_ms sh -c "python manage.py inspectdb -database=moodle > data_source/models_new.py"`

- Course
- User
- Context
- Role
- RoleAssignments

Two of the obtained model classes, `Course` and `RoleAssignments`, are enriched with an additional boolean attribute `ignore_activity` (defaulting to `True`) in order to provide an option to enable the assessment for specific `Course` instances and `User` instances related to these. Additionally, the `Course` class provides a `get_users_for_assessment()` method that returns the list of users for which assessment should take place for a specific `Course` instance, making use of the aforementioned boolean attribute.

The logic for the import of courses and users (and, just to mention it once, also of contexts, roles and role assignments) from the 'moodle' database into the project is defined inside the function `update_administration_data()` in the `tasks.py` file. For each of the model classes that inherit from the `AbstractMoodle` class inside the 'data_source' app's `models_abstract.py` file, the inheriting class of the 'administration' app's models is identified and subsequently synchronized with the inheriting class of the 'data_source' app's models. Also, instances of the 'administration' app's models that do not exist for the 'data_source' app's models are deleted. The function is made available to celery applying the `@shared_task` decorator.

Finally, the `views.py` file defines the views `CourseListView` and `RoleAssignmentsListView`, which present `Course` instances and `User` instances related to these to the user. These views show links that provide the option to enable or disable indicator calculations and construct assessments for these instances. To facilitate orientation within the user lists, pagination and a query function were added to the `RoleAssignmentsListView` that allow to filter for a user's id.

Two more views, `CourseStatusUpdateView` and `RoleAssignmentsStatusUpdateView`, serve as confirmation pages for the enabling/disabling of the assessment for an instance of `Course` or `RoleAssignments`.

5.2.4 'assessment' app

The configuration of indicators and constructs as well as their calculation and their storage to the database take place inside the app 'assessment'. Among the app's files and directories, the following will be discussed here in detail:

- `models/indicators.py`
- `views/indicators.py`
- `models/constructs.py`
- `views/constructs.py`
- `tasks.py`

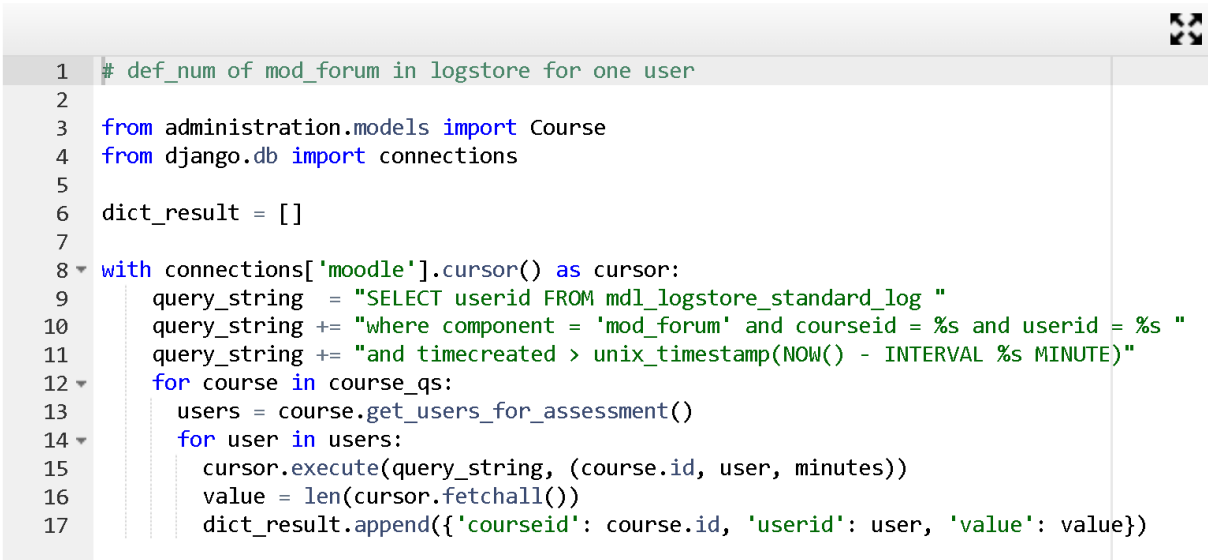
models/indicators.py

The file `indicators.py` in the directory `models/` of the app contains the model class `Indicator`, which serves to store indicators. It also contains a model class that is used to store the results of indicator calculations independent from a construct assessments: Those indicator results are saved as instances of the `IndicatorResult` model class.

Let's have a look at the `Indicator` class first. It defines the following attributes for its instances, whose relevance will be explained in the following paragraphs:

- `name`
- `column_label`
- `code`
- `minutes`
- `description`
- `DIFA_reference_id`
- `time_created`
- `last_time_modified`
- `schedule`
- `periodictask`

The `code` attribute contains the code that is executed in order to calculate indicator values. Figure 5.1 shows an example of the code entered for an `Indicator` instance in the `AceWidget` of the `IndicatorCreateView`.

A screenshot of a code editor window with a light gray background. The code is written in Python and is color-coded. It starts with a comment on line 1: `# def_num of mod_forum in logstore for one user`. Lines 3 and 4 import `Course` from `administration.models` and `connections` from `django.db`. Line 6 initializes `dict_result = []`. Line 8 starts a `with` block for a database cursor on the 'moodle' connection. Lines 9-11 build a SQL query string to select `userid` from `mdl_logstore_standard_log` based on `courseid`, `userid`, and a time filter. Line 12 starts a `for` loop over `course` in `course_qs`. Lines 13-17 calculate the number of users for each course within a specified time interval and append the result to `dict_result`.

```
1 # def_num of mod_forum in logstore for one user
2
3 from administration.models import Course
4 from django.db import connections
5
6 dict_result = []
7
8 with connections['moodle'].cursor() as cursor:
9     query_string = "SELECT userid FROM mdl_logstore_standard_log "
10    query_string += "where component = 'mod_forum' and courseid = %s and userid = %s "
11    query_string += "and timecreated > unix_timestamp(NOW() - INTERVAL %s MINUTE)"
12    for course in course_qs:
13        users = course.get_users_for_assessment()
14        for user in users:
15            cursor.execute(query_string, (course.id, user, minutes))
16            value = len(cursor.fetchall())
17            dict_result.append({'courseid': course.id, 'userid': user, 'value': value})
```

Figure 5.1: Sample indicator code

Source: Own conception.

The `Indicator` class provides a `calculate_result()` method that can be used whenever results need to be calculated for an instance of the `Indicator` class. When it is called, a specific calculation module is created or updated for this instance, making use of the above mentioned `code` attribute of the `Indicator` model.

The `calculate_result()` method accepts two additional parameters next to the `Indicator` instance: The first parameter `courses` expects a queryset of `Course` instances for which the results are supposed to be calculated, defaulting to all existing instances of the 'administration' app's `Course` class (except the Moodle site itself). From this queryset, the `Course` instances are removed for which assessment was disabled.

The second parameter `minutes` expects an integer defining a number of minutes that can be used inside the indicator calculation module to limit the time span to consider retrospectively when calculating the indicator results. This parameter defaults to `None`. This default is chosen only because it is not possible to call an instance method with a parameter set to an instance attribute: In the case that the calculation of indicator results does not happen as part of a construct assessment, the parameter cannot be set to the `Indicator` instance's attribute `minutes`. Therefore, the method contains a check whether the parameter is `None` and sets it to the instance's attribute `minutes` in that case.

Both parameters are handed over as variables `course_qs` and `minutes` to an indicator calculation module containing the code stored in the `code` attribute. Additionally, for each `Course` instance in `course_qs`, its `get_users_for_assessment()` method can be called to get the list of users that assessment results are to be calculated for (see 5.2.3). An example of how to use the `course_qs` and `minutes` variables and the `get_users_for_assessment()` method of the `Course` class can be seen in Figure 5.1, also showing that the indicator calculation module by convention must return a list of dictionaries called `dict_result`. Each of the dictionaries has the keys 'userid', 'courseid' and 'value'. After executing the code of the calculation module, the `calculate_result()` method returns this `dict_result` list.

The `Indicator` class also provides a `save_result()` method that is used to store an indicator instance's results as instances of the `IndicatorResult` class. The attributes of this class are instances of `Indicator`, `Course`, `User`, a timestamp and the corresponding result value. The `save_result()` method accepts the aforementioned parameter `courses` as well. It calls the method `calculate_result()` in order to calculate new results for the `Indicator` instance and then transforms these results into `IndicatorResult` instances. These instances are then stored to the database in bulk. Additionally, the `save_result()` method deletes those `IndicatorResult` instances from the database that contain `Course` and/or `User` instances for which an assessment has been disabled in the meantime.

views/indicators.py

The `views/indicators.py` file defines several views that present the `Indicator` instances and allow their configuration. First of all, the `IndicatorListView` presents an overview about all existing indicators.²² To create a new `Indicator` instance, the user enters its attributes `name`,

²² This view is also used to show all `Indicator` instances related to a specific instance of the `Construct` class. In that case, the view only shows `Indicator` instances standing in a `ConstructIndicatorRelation` to that `Construct` instance (see the following `views/constructs.py` subsection for more information).

column_label, code, minutes, description and DIFA_reference_id into a form embedded in the IndicatorCreateView. The attributes description and DIFA_reference_id are optional – they mainly serve to increase the semantic interoperability of the later results. When the Indicator instance is created, also a schedule and a periodic task are set up, whose id values are stored in the schedule and the periodictask attribute of the Indicator instance (see the following subsection on the tasks.py file). The schedule set up together with the Indicator instance (a crontab schedule) is initially set to * * * * *. The update of the related schedule is possible in the IndicatorScheduleUpdateView. The corresponding periodic task is initially deactivated. When the user activates the calculation of indicator values according to the configured crontab schedule, the IndicatorStatusUpdateView serves as a confirmation page.

The same form like in the IndicatorCreateView, but already bound to an existing instance of the Indicator class, is used in the IndicatorUpdateView to update the attributes of this instance. The IndicatorDeleteView serves as a confirmation page in case that the user demands to delete an Indicator instance from the database.

Two more views present the outcome of the calculate_result() and the save_result() method to the user. In both views, the Indicator instance's column_label attribute is used as a header when reporting the indicator results. The IndicatorCalculateListView delivers an ad-hoc report of the results of the calculate_result() function and can therefore serve the user to test whether the indicator code that was entered for the instance's code attribute in the IndicatorCreateView or IndicatorUpdateView works as expected.

The IndicatorResultsListView, on the other hand, shows all IndicatorResult instances in the default database whose indicator attribute equals a specific Indicator instance. Since the save_result() method stores IndicatorResult instances to the default database, the IndicatorResultsListView presents the outcomes of this method.

models/constructs.py

The constructs.py file in the directory models/ of the app contains the class Construct which serves to store constructs. It defines the following attributes for its instances, some of which are different to the attributes of the Indicator class. I will explain their relevance in this subsection.

- name
- indicators
- column_label
- minutes
- scaler
- description
- DIFA_reference_id
- time_created
- last_time_modified
- schedule
- periodictask

The `indicators` attribute allows to connect an instance of the `Construct` class with several instances of the `Indicator` class. The attribute is defined as a `ManyToManyField` and therefore allows one `Indicator` instance also to be connected to more than one `Construct` instance.

Moreover, the relationships between the `Indicator` and the `Construct` objects are mediated by the `ConstructIndicatorRelation` class which is referenced as `through` parameter of the `ManyToManyField`. This class additionally defines an additional attribute `weight` for the relationship between a `Construct` instance and an `Indicator` instance that is used to calculate the weighted average of indicator values in a construct assessment.

Another difference to the attributes of the `Indicator` class is the `scaler` attribute, which uses the `choices()` method of the `Scaler` class (also in `models/constructs.py`) to offer the user different transformation classes to use for preprocessing of indicator values in a construct assessment (it defaults to the option 'No scaler'). The offered preprocessing options are the ones documented for the scikit-learn preprocessing package (see 5.1.4).

The rest of the attributes of the `Construct` class is identical to the attributes of the `Indicator` class and will not be explained here again. Instead, we want to focus on the methods that the `Construct` class defines:

- `get_scaler()`
- `provide_indicator_results()`
- `calculate_result()`
- `save_result()`

Similar to the methods of the `Indicator` class, the methods of the `Construct` class accept a parameter `courses`.²³ It expects a queryset of `Course` instances for which construct results are to be provided, defaulting to all existing instances of the 'administration' app's `Course` class (except the Moodle site itself).

The names of the `calculate_result()` and the `save_result()` method are already known from the methods of the `Indicator` class. They in fact serve the same purpose. Nevertheless, since the calculation of a `Construct` instance's assessment results involves the calculation of values for all the `Indicator` instances related with that construct, they do not exactly work the same way.

But let's start with the two yet unknown methods. The `get_scaler()` method does exactly what its name suggests: It returns a `Scaler`, which is a scikit-learn-specific name for a transformer class or, in the context of this project, the class that will be used for preprocessing the indicators' values before they will be aggregated.

Inside the `provide_indicator_results()` method, the `get_scaler()` method is used to load the transformer class that was selected by the user in the `scaler` attribute of the respective `Construct` instance. If the user did not change the default option 'No scaler' for the `scaler` attribute in the `ConstructCreateView` or `ConstructUpdateView`, the indicators' values will not be preprocessed.

The `provide_indicator_results()` method generates indicator results as follows: First, the `Course` instances are removed from the queryset provided by the `courses` parameter for which assessment was disabled; the result is stored to the variable `course_qs`. Second, the queryset

²³ As stated in 4.4.2, this parameter could be used in the future to set up separate schedules for `Course` instances.

of the `Indicator` instances that are related to the `Construct` instance is retrieved and saved to the variable `indicators`. Third, using the `get_users_for_assessment()` method of each `Course` instance in the `course_qs` variable, entries in a dictionary are created for each user in each course. The result is the dictionary `indicator_results_dict`, whose keys are tuples of user id and course id. This dictionary serves the `provide_indicator_results()` method as a structure to store the preprocessed indicator results.

Now, for each `Indicator` instance in the `indicators` variable and for each `Course` instance in the `course_qs` variable, the `calculate_result()` method of the `Indicator` instance is called. In each call, its parameters `courses` and `minutes` equal to a queryset containing only the respective course and the value of the `minutes` attribute of the `Construct`. These calls each return the indicator results per indicator and course, which are then preprocessed with the previously loaded transformer class in the context of this one course and stored into the previously created structure `indicator_results_dict` as key-value pairs of the `Indicator` instance's attribute `column_label` and the respective preprocessed result. This dictionary is eventually returned by the `provide_indicator_results()` method.

The method `calculate_results()` uses the `provide_indicator_results()` method to obtain the indicator results in order to aggregate them into construct results. It therefore calls the `provide_indicator_results()` method and stores the returned dictionary of dictionaries in the variable `indicator_results`. Additionally, it obtains the weights assigned to the `Indicator` objects from the related `ConstructIndicatorRelation` instances and stores them into a variable `weights_dict` that uses the `Indicator` instances' `column_label` attribute as key to have the indicator weights ready when processing the indicator results.

Construct results are then calculated by going through the `indicator_results` dictionary of dictionaries and calculating for each key (a tuple of course id and user id) the weighted average of its value entries (key-value pairs of the `Indicator` instance's `column_label` and the corresponding value). The value of each of the key-value pairs is therefore multiplied with the weight obtained from the previously created `weights_dict`. The sum of all these value-weight products is stored as the value for the user in this course to the `construct_results` dictionary.

In difference to the previously described `calculate_results()` method of the `Indicator` class, the `calculate_results()` method of the `Construct` class returns two dictionaries: The `indicator_results` and the `construct_results`. Another difference becomes visible when we look at the classes that remain in the `models/constructs.py` file. In addition to the `Construct` class and the already mentioned `ConstructIndicatorRelation` class and the `Scaler` class, the file contains three more model classes which are used to store the results of the construct assessments to the database:

- `ConstructAssessment`
- `ConstructResult`
- `ConstructIndicatorResult`

These three classes are related to each other by foreign key relationships: Instances of the `ConstructIndicatorResult` class are related to `ConstructResult` instances through the `ForeignKey` field `constructresult`, and instances of the `ConstructResult` class are related to `ConstructAssessment` instances through the `ForeignKey` field `assessment`. An instance of the `ConstructAssessment` class represents a scheduled assessment of a `Construct` instance.

The idea behind these three classes is that one `ConstructAssessment` instance creates many `ConstructResult` instances that themselves are related to many `ConstructIndicatorResult` instances. The aforementioned structure allows the efficient retrieval of both a construct's latest assessment as well as its related results.

The `save_results()` method of the `Construct` class uses these three classes to store the results returned by the `calculate_results()` method. It first deletes those `ConstructResult` and `ConstructIndicatorResult` instances from the database that contain `Course` and/or `User` instances for which an assessment has been disabled since their creation. Then, the construct and indicator results are obtained by calling the `calculate_results()` method of the `Construct` class. After creating a new `ConstructAssessment` instance, the construct and indicator results are transformed into `ConstructResult` and `ConstructIndicatorResult` instances, relating lower-level with higher-level instances by using the `ForeignKey` fields `assessment` and `constructresult`, respectively. Eventually, all instances are stored to the database in bulk.

views/constructs.py

The `views/constructs.py` file defines the views that present the `Construct` instances and allow their configuration. It contains the following views:

- `ConstructListView`
- `ConstructCreateView`
- `ConstructUpdateView`
- `ConstructDeleteView`
- `ConstructScheduleUpdateView`
- `ConstructStatusUpdateView`
- `ConstructIndicatorWeightUpdateView`
- `ConstructIndicatorValuesView`
- `ConstructCalculateListView`
- `ConstructResultsListView`

Many of these views' names resemble the names of the views that were described in the subsection on the `views/indicators.py` file. As these views in fact do work exactly the same, I will go over them a little bit quicker here.

Among those almost identical views are the `ConstructListView`, the `ConstructCreateView`, the `ConstructUpdateView` and the `ConstructDeleteView`. Regarding these views I just want to mention that the form used to create/update `Construct` instances doesn't provide no `AcWidget` where the user has to enter executable code – instead, it contains a `ModelMultipleChoiceField` that allows to select the `Indicator` instances that are to be related with the construct.

Similarly to the `views/indicators.py` file as well, the `ConstructScheduleUpdateView` and the `ConstructStatusUpdateView` serve for the update of the related schedule and for the confirmation of the activation/deactivation of the related `periodictask` by the user. Identical to the creation of new `Indicator` instances, the related schedule is initially set to `* * * * *` and the related `periodictask` is initially deactivated when creating a new `Construct` instance.

New in the context of the `Construct` class is the `ConstructIndicatorValuesView`, which presents the outcome of the `provide_indicator_results()` method of a specific `Construct` instance. It can be used to check whether the selection of the transformation class in the `scaler` attribute of the `Construct` instance has the desired effects.

In close analogy to the `views/indicators.py` file, the `ConstructCalculateListView` and the `ConstructResultsListView` present the output of the `calculate_result()` and `save_result()` method to the user. Both views use the `column_label` attribute of the `Construct` instance as a header when reporting the indicator results. The `ConstructCalculateListView` can be used by the user as an ad-hoc report of the results for this `Construct` instance (for example after a readjustment of the weights). The `ConstructResultsListView` on the other hand shows the results from the database and can serve to inspect the results of the scheduled assessments.

As mentioned earlier in a footnote, the `IndicatorListView` from the `views/indicators.py` file is reused in the context of the `Construct` class. If the view can successfully retrieve a `Construct` object whose `id` attribute equals the keyword argument `construct_id` (obtained from the URL), the view only shows `Indicator` instances that are standing in a `ConstructIndicatorRelation` to that `Construct` instance. That makes it possible to display more details about the `Indicator` instances that are related to a specific `Construct` instance, as the `ModelMultipleChoiceField` in the `ConstructCreateView` and the `ConstructUpdateView` only shows the name of the related `Indicator` instances. It also shows each of the `ConstructIndicatorRelation` instances' `weight` attribute values.

The `ConstructIndicatorWeightUpdateView` allows the user to update these weights and to make sure that they sum up to one.

tasks.py

The `tasks.py` file defines two functions that serve a similar purpose for the `Indicator` and the `Construct` class:

- `save_indicator_results()`
- `save_construct_results()`

Both functions are prepended a `@shared_task` decorator in order to provide them as tasks to the celery service. Both functions use an argument `indicatorid` or `constructid`, respectively. In essence, they call a specific `Indicator` or `Construct` instance's `save_result()` method.

As already mentioned before, when an instance of the `Indicator` or `Construct` class is created, a `Schedule` instance arbitrarily set to `*****` and a `PeriodicTask` instance addressing the corresponding one of these two functions are set up using the `django-celery-beat` extension. Additionally, an argument value is added to the `PeriodicTask` instance which equals the `id` attribute of the just created `Indicator` or `Construct` instance. This argument value is used later on as the `indicatorid` or `constructid` argument when the `save_indicator_results()` or `save_construct_results()` function is called by celery.

The `id` attribute values of the `Schedule` and `PeriodicTask` instances, on the other hand, are added to the `Indicator` or `Construct` instance as values of its `schedule` and `periodictask` attributes. That makes it possible to update their configuration at a later stage, or to delete them as well if the user demands to delete the corresponding `Indicator` or `Construct` instance.

5.2.5 'export' app

The 'export' app exports the historical results of the indicator calculations and construct assessments, cleans up after their export and also provides results of the latest construct assessments through a REST API. Among the app's files and directories, the following will be discussed here in detail:

- models.py
- db.py
- tasks.py
- serializers.py
- views.py

The models.py file of the 'export' app contains the model classes that are used to store the historical results of the indicator calculations and construct assessments in the database available through the 'cassandra' connection. The model classes in this file inherit from the DjangoCassandraModel class which is provided by the django_cassandra_engine extension (see 5.1.3). The file defines the following model classes:

- IndicatorResultExport
- ConstructAssessmentExport
- ConstructResultExport
- ConstructIndicatorResultExport

Their attributes are similar to the IndicatorResult, ConstructAssessment, ConstructResult and ConstructIndicatorResult classes already known from the models/indicators.py and the models/constructs.py file of the 'assessment' app. The only difference to these is that each of the corresponding classes in the 'export' app's models.py file lacks the exported attribute. This is because the exported attribute is used to flag the results that were exported already, which is why it would always be True in the exported results and therefore not carry any information.

Unlike the other apps, the 'export' app has an additional db.py file. This is because the settings.py file of the 'app' project does not obligatorily establish a 'cassandra' connection at startup. Therefore, the code in this file serves to add a 'cassandra' connection to Django's connections dictionary in case it does not exist there yet.

The 'export' app's tasks.py file defines the four functions that export the historical results to the 'cassandra' database and clean up after their export. All of the four functions are prepended a @shared_task decorator to make them available to the celery service:

- export_indicator_results()
- cleanup_exported_indicator_results()
- export_construct_assessments()
- cleanup_exported_construct_assessments()

Similarly to the functions in the 'assessment' app's tasks.py file, these functions serve a similar purpose for two groups of classes: On the one hand the IndicatorResult class and on the other the classes ConstructAssessment, ConstructResult and ConstructIndicatorResult.

The functions `export_indicator_results()` and `export_construct_assessments()`, which do not accept any arguments, serve to identify the instances of the `IndicatorResult` class (or the `ConstructAssessment`, `ConstructResult` and `ConstructIndicatorResult` instances, respectively) whose attribute `exported` is set to `False`. For these instances, they create instances of the before mentioned class `IndicatorResultExport` (or, respectively, instances of the before mentioned classes `ConstructAssessmentExport`, `ConstructResultExport`, and `ConstructIndicatorResultExport`) in the 'cassandra' database. Eventually, they set the `exported` attribute for all of the just exported instances to `True`.

The flipped value of the `exported` attribute then becomes relevant for the two functions `cleanup_exported_indicator_results()` and `cleanup_exported_construct_assessments()`, which identify those `IndicatorResult` instances (or those instances of `ConstructAssessment`, `ConstructResult` and `ConstructIndicatorResult`, respectively) which are already flagged as `exported`. They delete those instances from the default database.

There is a small but important difference between `cleanup_exported_indicator_results()` and `cleanup_exported_construct_assessments()` besides the fact that one cleans up the exported instances of just one model and the other one the instances of three models: While `cleanup_exported_indicator_results()` deletes all identified instances of `IndicatorResult` flagged by the `exported` attribute, the `cleanup_exported_construct_assessments()` function tags for each `Construct` instance represented by the existing `ConstructAssessment` instances the latest `ConstructAssessment` instance in terms of their `timestamp` attribute beforehand. These instance and their related `ConstructResult` and `ConstructIndicatorResult` instances are then excepted from the deletion. That way, the latest set of assessments results is always kept in the default database for each `Construct` instance.

The `serializers.py` file comprises two classes that are used to provide these assessment results in JSON format:

- `ConstructIndicatorResultSerializer`
- `ConstructResultSerializer`

Both classes inherit from the `ModelSerializer` class which is provided by the Django REST framework package. Together they serialize the instances of the 'assessment' app's model classes `ConstructResult` and `ConstructIndicatorResult`.

In order to represent the nested relationship²⁴ between the two underlying model classes, the `ConstructIndicatorResultSerializer` class is embedded as the field `measures` in the `ConstructResultSerializer` class. This approach was inspired by feedback following the interim presentation of the thesis project which suggested that it would be helpful to deliver the results of the construct assessments together with their underlying indicator values. Figure 5.2 shows a draft JSON response provided by the supervisor as feedback to the presentation.

Also inspired by this draft was the inclusion of metadata. Fortunately, the design of the project allows it to make use of the relationships between the model classes in order to enrich the serialized output with additional meta information like the construct's and the indicators' name and description and the timestamp of the assessment.

²⁴ <https://www.django-rest-framework.org/api-guide/relations/#nested-relationships>

```
data = {
    "user_id": 1,
    "course_id": 1,
    "construct_name": "learning_engagement",
    "analysis_timestamp": 202008181513,
    "description": "....",
    "result": 0.87,
    "measures": [
        {
            "measure_id": 1001,
            "measure_name": "Forum social",
            "description": "This measure is ...",
            "result": 0.1
        }
    ]
}
```

Figure 5.2: Draft JSON response

Source: Adapted from Skype conversation with George P. Ciordas-Hertel (supervisor)

The `views.py` file defines a class `ConstructResultsAPIView` that inherits from the `APIView` class provided by the Django REST framework package. It uses the just presented class `ConstructResultSerializer` to provide a JSON response for the REST API.

The `ConstructResultsAPIView` first builds a queryset that contains those `ConstructResult` instances that are supposed to be serialized. It therefore creates an empty `ConstructResult` queryset and then identifies the relevant `ConstructResult` instances by going through all existing `Construct` instances one after another. For each one of them, it determines the latest `ConstructAssessment` instance in terms of its timestamp attribute and adds all `ConstructResult` instances to the queryset whose `assessment` attribute equals the determined most recent `ConstructAssessment` instance. The `ConstructResultSerializer` class is then initialized with the obtained `ConstructResult` queryset. Finally, the serializer object's `data` attribute is returned to the client, applying the `Response` object of the Django REST framework package. That way, the JSON response is provided when the API endpoint is called.

5.3 Project workflow

In this last section, I will briefly explain how the project is meant to work and what steps need to take place one after another for the project to deliver the results of a construct assessment. This is supposed to facilitate the understanding of how the before mentioned components play together and also supposed to prepare an understanding for the paradigmatic implementation of scheduled assessments of the construct of Learning Engagement in chapter 6.

1. First of all, the 'administration' app needs to load data from the database of the Moodle LMS using the 'moodle' connection of the Django project. This happens triggered by a schedule that is configured to run the process repeatedly. Since the assessment is initially disabled for all newly imported courses and users, the user needs to enable the assessment for at least one course, and in this course for at least one user in order to obtain results from the scheduled construct assessments later.
2. The user needs to create `Indicator` instances. Especially, the user needs to provide the Python code for each indicator. The code should make use the `courses_qs` and `minutes` variables and to store the results in a variable `dict_result` as a list of dictionaries that each contain the keys 'courseid', 'userid' and 'value' (cf. Figure 5.1). The user should test the calculation of each indicator's values by triggering an ad-hoc report of the indicator values.
3. The user needs to create a `Construct` instance and assign indicators to it. Moreover, the user needs to provide values for the `minutes` and the `scaler` attribute of the construct.
4. The user needs to assign construct-specific weights to the assigned indicators. Afterwards, the user should test the calculation of the transformed indicator values for `Construct` by triggering a report of the preprocessed indicator values. Additionally, the user should test the calculation of the construct's values by triggering an ad-hoc report of the construct values.
5. The user should configure the construct-specific schedule. Additionally, the user needs to enable the construct-specific periodic task. Then, the user has to wait for the scheduled time to arrive.
6. The user should now see the results of the construct assessment by opening the report of construct assessment results. External applications should be able to obtain the results of the construct assessment by calling the REST API. After an export of the results has taken place, the results of the construct assessment become available in the Cassandra database. After a cleanup has taken place, the user can no longer see the results of the construct assessment by opening the report of construct assessment results.

6 Evaluation

This chapter presents the evaluation of the implemented system. I start this chapter by returning to the acceptance criteria of chapter 4 and by checking for each of those criteria whether the underlying feature was implemented as expected. In the second section of the chapter I will exemplarily implement scheduled assessments of the construct of Learning Engagement. That second part of the evaluation is supposed to function as a user guide to the solution as well. Finally, I will review the evaluation and talk about its limitations.

6.1 Reconsidering features and acceptance criteria

In section 4.3 I introduced acceptance criteria that will be used here to evaluate whether the implemented project has the features that were presented in section 4.2. I will go through the features and their acceptance criteria one by one in order to systematically check whether the project meets the requirements.

Feature	Acceptance criterion
Pending assessments identification	The project provides the option to set whether or not a course, and in this course, a user will be object to future assessments. Scheduled assessments are performed automatically only for those courses and users that were previously selected.

Table 6.1: Acceptance criteria for 'Pending assessments identification' (4.2.1)

Source: own conception

The first feature 'Pending assessments identification', depicted in Table 6.1, was entirely implemented in the 'administration' app. This app's model classes continuously replicate the data of the Moodle database about courses, users and their relationship. It adds an attribute to two of these classes in order to flag courses and users as designated for indicator calculations and construct assessments. Assessments are initially disabled for all courses and all their users, so that scheduled assessment do not deliver any results initially without further configuration. For this configuration, the 'administration' app's views allow to enable (and disable again) the assessments for courses and their users. This satisfies the acceptance criteria of the feature and it is therefore considered as successfully implemented.

Feature	Acceptance criteria
Assessment scheduling	Construct assessments are performed automatically following a configurable schedule without further user input being necessary. The scheduling of the construct assessments can be turned on and off.

Table 6.2: Acceptance criteria for 'Assessment scheduling' (4.2.2)

Source: own conception

Table 6.2 presents the feature 'Assessment scheduling'. This feature was implemented inside the 'assessment' app's views/indicators.py, views/construct.py and tasks.py file and uses as well the more general functionality implemented in the 'app' project's celery.py file. The scheduling of construct assessments is already arranged inside the app's views when a new construct is created. Based on the task in the tasks.py file, a separate crontab schedule and a separate periodic task are automatically created for this construct. After the construct has been successfully configured itself, the crontab schedule can be individually configured and the periodic task can be turned on and off. According to this configuration, construct assessments are then triggered automatically as celery tasks at the specified time only if the corresponding periodic task is enabled. Hence, the acceptance criteria of the feature are met. The option to individually schedule each indicator and construct surpasses the feature's requirements.

Feature	Acceptance criteria
Assessment performance	Assessments can be started by calling a method and the aggregated values for the construct / original values for the indicator are returned according to the pending assessments identification. All assessment results are returned together in one single data structure.

Table 6.3: Acceptance criteria for 'Assessment performance' (4.2.3)

Source: own conception

The feature 'Assessment performance' and its acceptance criteria are shown in Table 6.3. The assessment mechanism was implemented in the models/indicators.py and models/constructs.py files of the 'assessment' app. For indicator values, the `calculate_result()` method of the `Indicator` class alone delivers a dictionary of results after running the code that was entered for the respective indicator by the user. For constructs, the `calculate_result()` method of the `Construct` class delivers two dictionaries: One that contains the assessment results and one that additionally contains all indicator values that were used to calculate those construct values. In both cases, the methods deliver values only for those courses and users in these courses for which assessment was enabled before according to the 'Pending assessments identification' feature. It can therefore be said that the requirements of the feature are met, in the case of the construct assessments they were exceeded as additional information about the assessment is provided together with the results.

Feature	Acceptance criteria
Results storage	Results of construct assessments and indicator calculations can be viewed when connecting to the database. Ideally, the assessment results already existing for a course or user for whom assessment was disabled are deleted from the database.

Table 6.4: Acceptance criteria for 'Results storage' (4.2.4)*Source:* own conception

Table 6.4 exhibits the acceptance criteria of the feature 'Results storage'. This feature was implemented in the `models/indicators.py` and `models/constructs.py` files of the 'assessment' app. The `IndicatorResult` model class stores the results of the indicator calculations that were triggered separately. The `ConstructIndicatorResult` model class, on the other hand, stores the results of the indicator calculations that were part of a construct assessment. The aggregated construct results are stored by the `ConstructResult` model class and the construct assessment itself is stored as an instance of the `ConstructAssessment` model class. When connecting to the SQLite database, the indicator results can be viewed as rows of the tables `assessment_indicatorresult` or `assessment_constructindicatorresult`, respectively. The construct results and the assessment can be found in the tables `assessment_constructresult` and `assessment_constructindicatorresult`. The `calculate_result()` methods of the `Indicator` and the `Construct` class use these model classes to store the results of construct assessments and indicator calculation and also delete the already existing results of courses and/or users for which the assessment was disabled in the meantime. In that sense, the implementation fully complies with the acceptance criteria of this feature.

Feature	Acceptance criteria
Results report	Course- and user-specific assessment results including a timestamp of the assessment are reported after an assessment has taken place. Ideally, an ad-hoc assessment can be started and the results be viewed without saving to the database.

Table 6.5: Acceptance criteria for 'Results report' (4.2.5)*Source:* own conception

The acceptance criteria of the feature 'Results report' are described in Table 6.5. The feature was implemented in the `views/indicator.py` and `views/constructs.py` files of the 'assessment' app. The `IndicatorResultsListView` and the `ConstructResultsListView` report the results of previous indicator calculations or construct assessments, respectively, to the user, including a timestamp for each result. Additionally, the `IndicatorCalculateListView` and the `ConstructCalculateListView` serve as ad-hoc report of indicator or construct values, allowing to 'stage' the results that would be generated in a scheduled indicator calculation or construct assessment. Even an ad-hoc report of the transformed indicator values is provided by the `ConstructIndicatorValuesView`. The implementation thus match the acceptance criteria.

Feature	Acceptance criteria
Results emission	Assessment results are exported to and persisted in an external database. If the external database is unavailable, this does not affect the assessment performance. In that case, scheduled assessments continue to take place. Assessment results are not exported only until the external database becomes available again.

Table 6.6: Acceptance criteria for 'Results emission' (4.2.6)

Source: own conception

Finally, Table 6.6 pictures the acceptance criteria of the feature 'Results emission'. This was implemented in the files `db.py`, `models.py` and `tasks.py` of the 'export' app. The function `export_indicator_results()` exports the results as instances of the `IndicatorResultExport` model class to the 'cassandra' database. In the case of construct assessment results, the function `export_construct_assessments()` exports the results as `ConstructAssessmentExport`, `ConstructResultExport` and `ConstructIndicatorResultExport` instances to the 'cassandra' database. If the Cassandra database is not available, the application can still perform assessments and store them to the default database. Even if the 'cassandra' connection is unknown because Cassandra was unavailable at the startup of the project, the project retries to establish a 'cassandra' connection every time the `export_construct_assessments()` or the `export_construct_assessments()` function are called. The functions `cleanup_exported_indicator_results()` and `cleanup_exported_construct_assessments()` allow to delete the results that were already exported from the default database in order to keep the size of the database within a certain limit. Regarding the export to an external database, the implemented project fulfils the acceptance criteria of this feature.

Since the epic of the feature 'Results emission' was called 'Provision of results', it should be mentioned here that an additional way to provide the results of the construct assessments was implemented in the `serializers.py` and the `views.py` file of the 'export' app. The class `ConstructResultsAPIView` provides the results of the latest assessment of a construct in JSON format, including the underlying indicator values and additional metadata. This enables external applications to retrieve the current values of all constructs by calling a REST API. Regarding the provision of current construct results in JSON format to external applications, the implemented project stands out against the acceptance criteria.

6.2 Assessment of Learning Engagement (User guide)

In this section, I exemplarily implement scheduled assessments of the construct of Learning Engagement.

6.2.1 Software installation, system startup and provision of courses and users

First of all, the repository needs to be cloned using the command

```
git clone https://github.com/jakobschroeber/bachelor_thesis.git
```

To build the project container and to run all docker containers, the following command needs to be run in the installation directory:

```
docker-compose up
```

Now, we have seen a number of commands are executed. We need to wait for the Moodle instance to become available at <http://localhost:10080> (this may take up to 30 minutes the first time you run the docker-compose setup because Moodle installation takes some time).

Meanwhile, once we see that the `assessment_ms` container is up, we can open one more terminal window where we change to the installation directory and run the commands

```
docker exec assessment_ms sh -c "python manage.py makemigrations"
docker exec assessment_ms sh -c "python manage.py migrate"
docker exec assessment_ms sh -c "python manage.py sync_cassandra"
```

This creates and executes the database migrations necessary for the Django ORM to work (this required only once after fresh installation). Also right after a fresh installation, we need to import initial data to the database using the command

```
docker exec assessment_ms sh -c "python manage.py loaddata initial_data.json"
```

This will provide data for a few basic background tasks. We can leave this window open, we can use it later to stop the application. We would then just need to enter the command

```
docker-compose down
```

Additionally, in order to start a celery worker and the task scheduler, we need to open two more terminal windows where we also change to the installation directory and run the following two commands (in two different terminal windows):

```
docker exec assessment_ms sh -c "celery -A app worker -l info --concurrency=1"
docker exec assessment_ms sh -c "celery -A app beat -l info --scheduler
    django_celery_beat.schedulers:DatabaseScheduler"
```

Once we see the Moodle welcome page at <http://localhost:10080>, we can log in using the default MOODLE_USERNAME 'user' and the default MOODLE_PASSWORD 'bitnami' in order to populate the Moodle instance with courses. For this purpose, we create a backup file of both the two courses "Psychology in cinema" and "Celebrating Cultures" that are part of the Mount Orange school demo¹ and restore them inside our Moodle instance (I will not go into details here on how to achieve that).

Now, we have courses, but no users yet that take any action. That's why we create a few users newuser, newuser2 and newuser3 in Moodle that we can use afterwards to test the construct assessments. We still have to log in with every user (best to use a single browser instance for each) and self-enrol each of them in the two courses.

6.2.2 Activation of assessment for courses and their users to the project

When we access the project at <http://localhost:8000/administration/courses/>, we should see that the the two courses are there and the users are also listed in these two courses (Figure 6.1). It may take a minute until the background task imported the courses.

List of all courses

ID	Full name	Short name	Start date	End date	Status	Actions
4	Psychology in Cinema	Psych Cine	2014-07-16	1970-01-01	Off	Enable assessment Show assigned users
5	Celebrating Cultures	Celebrating Cultures	2020-09-22	1970-01-01	Off	Enable assessment Show assigned users

Figure 6.1: Courses listed after import

Source: Own conception

We can now enable the assessment for one of the courses. Let's pick the course "Psychology in cinema". After a click on the link "Enable assessment" next to course, we get to a confirmation page for the activation of this course for assessment (Figure 6.2).

Enable assessment for course

Enable assessment for course 4 (Psychology in Cinema)?

Do it!

Figure 6.2: Confirmation page for course activation

Source: Own conception

¹ <https://school.moodledemo.net/>

We also need to enable the assessment for our three users. The status of the course "Psychology in cinema" and the three users newuser, newuser2 and newuser3 should be 'On' afterwards (Figure 6.3).

List of all users in course 4 (Psychology in Cinema)

User ID	User name	Assignment	Role	Status	Actions
4	heatherreyes169	17	editingteacher	Off	Enable assessment
5	teacher	19	editingteacher	Off	Enable assessment
17	student	18	student	Off	Enable assessment
20	newuser	20	student	On	Disable assessment
21	newuser2	21	student	On	Disable assessment
37	newuser3	44	student	On	Disable assessment

« First page Previous page - Page 1 of 1 - Next page Last page »

Figure 6.3: Users listed after import

Source: Own conception

6.2.3 Configuration of Learning Engagement indicators

In the next step, we need to add a few indicators. In order to do that, we change to the list of indicators (available at <http://localhost:8000/assessment/indicators/> or simply by clicking on 'Indicators' in the navigation).

An empty indicator list appears, we see only the link "Create new indicator" (Figure 6.4).

If we follow that link, we get to a page with an empty indicator creation form (Figure 6.5).

In this form, we enter the indicator name, a column label, the number of minutes to consider retrospectively for the indicator and optionally a description and the DIFA ID). When we enter the Python code to execute when indicator values are needed, we remember to use the `courses_qs` and `minutes` variables to provide the courses and to limit the query results to variable time frame. Also, we store the results in a variable `dict_result` as a list of dictionaries that each contain the keys 'courseid', 'userid' and 'value' (Figure 6.6).

We get redirected to the indicator list, that now shows one indicator. We also see that the indicator has a schedule that is set to '* * * * *' and that it is not enabled for task scheduling (Figure 6.7).

Now, we can check the indicator values by clicking on the link "Calculate ad-hoc values of indicator" next to our new indicator in the list. Of course, it will be more interesting if our users have been active in the last minutes (we have set the relevant number of minutes in the `minutes` attribute of the indicator). We see that the ad-hoc report only returns values for the enabled course and the enabled users (Figure 6.8).

List of all indicators

[Create new indicator](#)

Figure 6.4: Empty indicator list

Source: Own conception

Create new indicator

```
1 # Python code for indicator calculation
2
3
```

Figure 6.5: Empty indicator creation form

Source: Own conception

Number of Forum Access

num(accesses to foren)

```

1  # def_num of mod_forum in logstore for one user
2
3  from administration.models import Course
4  from django.db import connections
5
6  dict_result = []
7
8  with connections['moodle'].cursor() as cursor:
9      query_string = "SELECT userid FROM mdl_logstore_standard_log "
10     query_string += "where component = 'mod_forum' and courseid = %s "
11     query_string += "and userid = %s and timecreated > unix_timestamp(NOW() - INTERVAL %s MINUTE)"
12     for course in course_qs:
13         users = course.get_users_for_assessment()
14         for user in users:
15             cursor.execute(query_string, (course.id, user, minutes))
16             value = len(cursor.fetchall())
17             dict_result.append({'courseid': course.id, 'userid': user, 'value': value})

```

10

from Aluja-Banet et al. (2018)

1018

save

Figure 6.6: Filled indicator creation form

Source: Own conception

It would now be possible to configure the indicator's schedule for indicator calculations by clicking on the link "Update schedule" and to turn on the task scheduling for this indicator by clicking on the link "Enable assessment". Results would then be shown in the report accessible by the link "Show results from database" as soon as they become available in the database.

Instead, we will proceed to the construct configuration. We will use the task scheduling of the construct, which will in turn trigger the calculation of indicator values (by using the `minutes` attribute of the construct).

List of all indicators

Indicator name	Column label	DIFA ID	Created	Last modified	Schedule Enabled	Actions
4 Number of Forum Access	num(accesses to foren)	1018	Sept. 22, 2020, 4:55 p.m.	Sept. 22, 2020, 5:21 p.m.	* * * * * False	Update indicator Update schedule Enable assessment Delete indicator Calculate ad-hoc values of indicator Show results from database

[Create new indicator](#)

Figure 6.7: Indicator list with one indicator

Source: Own conception

Current values of indicator 4 (Number of Forum Access)

Course ID	User ID	num(accesses to foren)
4	20	4
4	21	2
4	37	2

Figure 6.8: Indicator ad-hoc values report

Source: Own conception

6.2.4 Configuration of the Learning Engagement construct

In this step, we will set up the construct. In order to do that, we change to the list of constructs (available at <http://localhost:8000/assessment/constructs/> or simply by clicking on 'Constructs' in the navigation). An empty construct list appears, as we had seen it already for the indicators. We only see the link "Create new construct" (analog to Figure 6.4).

If we follow that link, we get to a page with an empty construct creation form (Figure 6.9).

In this form, we enter the construct name, a column label, the number of minutes to consider retrospectively for the construct and optionally a description and the DIFA ID). So far, it is very similar to the creation of a construct. But instead of entering Python code, we select from the available indicators. Additionally, we should select a scaler from the dropdown list (Figure 6.10).

We get redirected to the construct list, that now shows one construct. We also see that, similar to the indicator list, the new construct has a schedule that is set to '* * * * *' and that it is not enabled for task scheduling (Figure 6.11).

Create new construct

- ☐ Number of Forum Access

Figure 6.9: Empty construct creation form

Source: Own conception

- ☒ Number of Forum Access

Figure 6.10: Filled construct creation form

Source: Own conception

List of all constructs

Construct name	Column label in database results	DIFA ID	Created	Last modified	Schedule Enabled	Actions
2 Learning Engagement	LE		Sept. 22, 2020, 6:29 p.m.	Sept. 22, 2020, 6:29 p.m.	* * * * * False	Update construct Update schedule Enable assessment Delete construct Show assigned indicators and weights Calculate values of assigned indicators Calculate ad-hoc values of construct Show results from database

[Create new construct](#)

Figure 6.11: Construct list with one construct

Source: Own conception

6.2.5 Assignment of weights to the indicators

The next step is to check the assigned indicators and their weights. By clicking on the link "Show assigned indicators and weights" we get to a view that we already know from the indicator list – just that the indicators in this indicator list are filtered for a relationship with our construct (Figure 6.12).

List of indicators assigned to construct 2 (Learning Engagement)

Indicator name	Column label	DIFA ID	Created	Last modified	Weight	Actions
4 Number of Forum Access	num(accesses to foren)	1018	Sept. 22, 2020, 4:55 p.m.	Sept. 22, 2020, 5:21 p.m.	0.0	Update indicator Delete indicator Calculate ad-hoc values of indicator Show results from database

[Create new indicator](#) [Adjust indicator weights](#)

Figure 6.12: Indicator list filtered for construct

Source: Own conception

We can also see that compared with the indicator list before, the indicators here have different columns – The columns 'Schedule' and 'Enabled' are not there, but instead there is a column 'Weight'. The value is initially set to '0.0', which we can change by clicking on the link "Adjust indicator weights" below the table.

Following that link, we get a page showing the list of indicators related to our construct with a form field that allows to change the weight of the indicators. An additional field allows to calculate the sum of all indicator weights when the button "show sum" is clicked (Figure 6.13). This is relevant because we can only successfully submit the form with the "Save" button if the weights sum up to 1. Otherwise, we will see the message "The weights do not sum up to 1.0".

Adjust weights of indicators assigned to construct 2 (Learning Engagement)

Indicator name	DIFA ID	Weight
4 Number of Forum Access	1018	<input type="text" value="1,0"/>
		<input type="text" value="1"/>

Figure 6.13: Indicator weights update list for construct

Source: Own conception

If we now get redirected to the indicator list filtered for a relationship with our construct, we see that the weights were updated. We can get back to the construct list by clicking on 'Constructs' in the navigation.

We can now check the transformed indicator values by clicking on the link "Calculate values of assigned indicators" next to our new construct in the list. Of course, it will be more interesting if our users have been active in the last minutes (we have set the relevant number of minutes in the `minutes` attribute of the construct). Again, we see that the ad-hoc report only returns values for the enabled course and the enabled users. We can also observe that the indicator values were scaled, in this case to a range between 0 and 1 using the `MinMaxScaler` (Figure 6.14).

Current indicator values of construct 2 (Learning Engagement)

Course ID	User ID	num(accesses to foren)
4	20	0.9999999999999998
4	21	0.857142857142857
4	37	0.0

Figure 6.14: Transformed indicator values for construct

Source: Own conception

We can check as well the construct values by clicking on the link "Calculate ad-hoc values of construct" next to our new construct in the list (again, the number of minutes set in the `minutes` attribute of the construct will be used). We see that the ad-hoc report returns values only for the enabled course and the enabled users (Figure 6.15).

Current values of construct 2 (Learning Engagement)

courseid	userid	LE
4	20	1.0
4	21	0.875
4	37	0.0

Figure 6.15: Construct ad-hoc values report

Source: Own conception

6.2.6 Scheduling of assessments

Finally, we can set the schedule for the construct. In order to do so, we click on the link "Update schedule" next to our construct in the construct list. We get to a page that allows us to set the crontab.² Let's say we want construct assessments to be performed every five minutes (Figure 6.16).

Update schedule of construct 2 (Learning Engagement)

Cron Minutes to Run. Use "*" for "all". (Example: "0,30")

Cron Hours to Run. Use "*" for "all". (Example: "8,20")

Cron Days Of The Week to Run. Use "*" for "all". (Example: "0,5")

Cron Days Of The Month to Run. Use "*" for "all". (Example: "1,15")

Cron Months Of The Year to Run. Use "*" for "all". (Example: "0,6")

Cron Timezone: Timezone to Run the Cron Schedule on. Default is UTC.

Figure 6.16: Construct schedule configuration

Source: Own conception

When we are redirected to the construct list, we see that the value in the 'Schedule' column was updated for our construct. We now only need to turn the task scheduling on. To do that, we need to click on the link "Enable assessment" and confirm in the following page that we want to activate the scheduled assessments for our construct (Figure 6.17). It typically takes some seconds until celery-beat realizes that the schedule and the periodic task were updated.

² <https://crontab.guru/> can be of great help

Enable assessment for construct

Enable assessment for construct 2 (Learning Engagement)?

Do it!

Figure 6.17: Construct assessment activation

Source: Own conception

6.2.7 Analysis of results

If we now change to the report of construct assessment results (clicking on the link "Show results from database") next to our construct in the construct list, we should see after five minutes the results of the construct assessment (of course we have to update the page from time to time). In our example, we see that assessment results are created every five minutes (Figure 6.17).

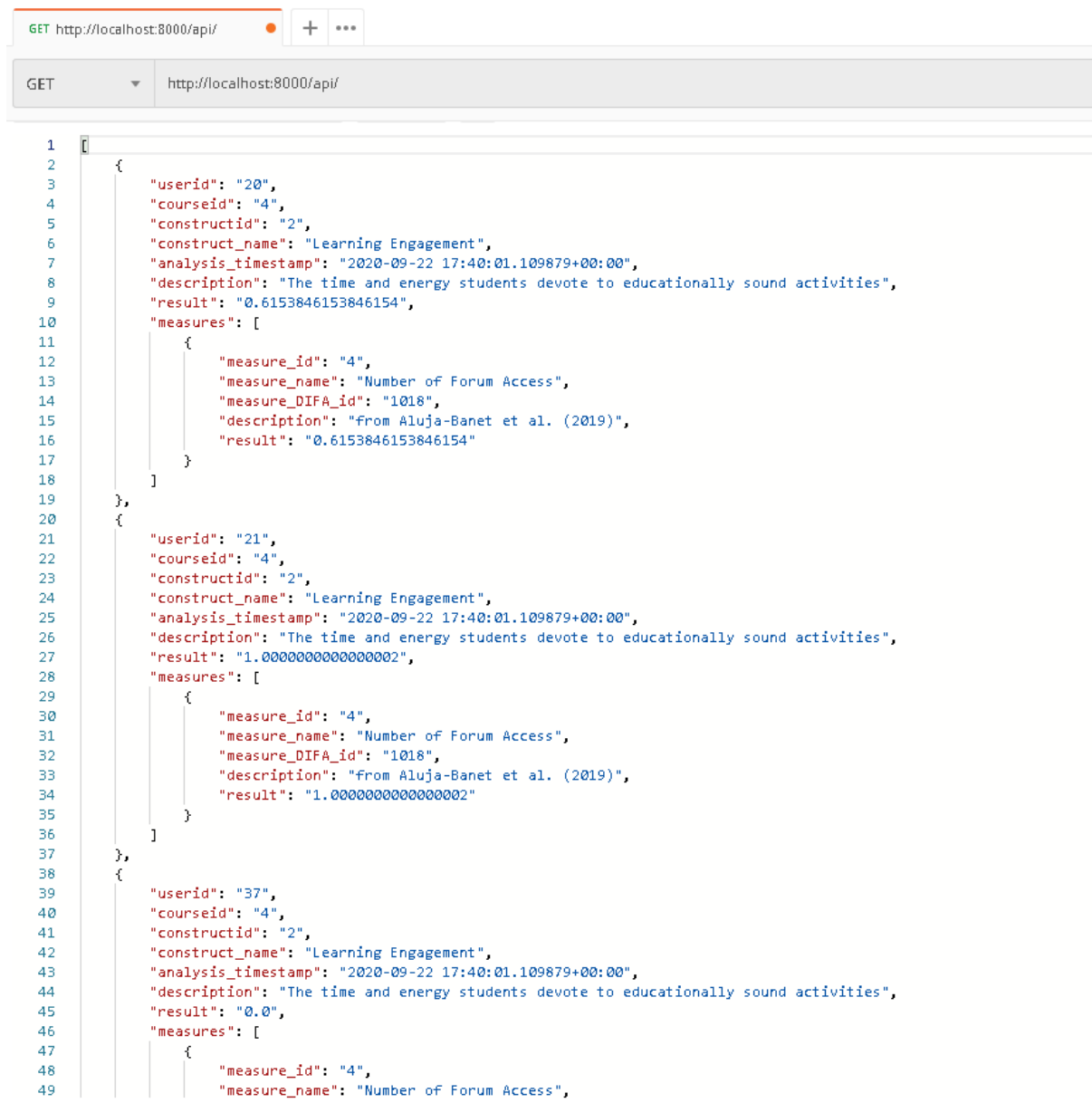
Database results of construct 2 (Learning Engagement)

Course ID	Course full name	User ID	Username	LE	Created	Exported
4	Psychology in Cinema 20	newuser	0.0		Sept. 22, 2020, 7:35 p.m.	False
4	Psychology in Cinema 21	newuser2	0.0		Sept. 22, 2020, 7:35 p.m.	False
4	Psychology in Cinema 37	newuser3	0.0		Sept. 22, 2020, 7:35 p.m.	False
4	Psychology in Cinema 20	newuser	0.6153846153846154		Sept. 22, 2020, 7:40 p.m.	False
4	Psychology in Cinema 21	newuser2	1.0000000000000002		Sept. 22, 2020, 7:40 p.m.	False
4	Psychology in Cinema 37	newuser3	0.0		Sept. 22, 2020, 7:40 p.m.	False

Figure 6.18: Construct assessment results report

Source: Own conception

Also, external applications should be able to obtain the latest results of the construct assessment by calling the REST API at <http://localhost:8000/api/> (Figure 6.17).



```

1  {
2    {
3      "userid": "20",
4      "courseid": "4",
5      "constructid": "2",
6      "construct_name": "Learning Engagement",
7      "analysis_timestamp": "2020-09-22 17:40:01.109879+00:00",
8      "description": "The time and energy students devote to educationally sound activities",
9      "result": "0.6153846153846154",
10     "measures": [
11       {
12         "measure_id": "4",
13         "measure_name": "Number of Forum Access",
14         "measure_DIFA_id": "1018",
15         "description": "From Aluja-Banet et al. (2019)",
16         "result": "0.6153846153846154"
17       }
18     ]
19   },
20   {
21     "userid": "21",
22     "courseid": "4",
23     "constructid": "2",
24     "construct_name": "Learning Engagement",
25     "analysis_timestamp": "2020-09-22 17:40:01.109879+00:00",
26     "description": "The time and energy students devote to educationally sound activities",
27     "result": "1.0000000000000002",
28     "measures": [
29       {
30         "measure_id": "4",
31         "measure_name": "Number of Forum Access",
32         "measure_DIFA_id": "1018",
33         "description": "From Aluja-Banet et al. (2019)",
34         "result": "1.0000000000000002"
35       }
36     ]
37   },
38   {
39     "userid": "37",
40     "courseid": "4",
41     "constructid": "2",
42     "construct_name": "Learning Engagement",
43     "analysis_timestamp": "2020-09-22 17:40:01.109879+00:00",
44     "description": "The time and energy students devote to educationally sound activities",
45     "result": "0.0",
46     "measures": [
47       {
48         "measure_id": "4",
49         "measure_name": "Number of Forum Access",

```

Figure 6.19: Construct assessment latest results in REST API

Source: as presented in Postman application

Since we made construct's `minutes` attribute to 15, we see that the values go back to zero after 15 minutes if there is no further user activity (Figure 6.20).

Database results of construct 2 (Learning Engagement)

Course ID	Course full name	User ID	Username	LE	Created	Exported
4	Psychology in Cinema 20	newuser	0.0		Sept. 22, 2020, 7:35 p.m.	False
4	Psychology in Cinema 21	newuser2	0.0		Sept. 22, 2020, 7:35 p.m.	False
4	Psychology in Cinema 37	newuser3	0.0		Sept. 22, 2020, 7:35 p.m.	False
4	Psychology in Cinema 20	newuser	0.6153846153846154		Sept. 22, 2020, 7:40 p.m.	False
4	Psychology in Cinema 21	newuser2	1.0000000000000002		Sept. 22, 2020, 7:40 p.m.	False
4	Psychology in Cinema 37	newuser3	0.0		Sept. 22, 2020, 7:40 p.m.	False
4	Psychology in Cinema 20	newuser	0.6153846153846154		Sept. 22, 2020, 7:45 p.m.	False
4	Psychology in Cinema 21	newuser2	1.0000000000000002		Sept. 22, 2020, 7:45 p.m.	False
4	Psychology in Cinema 37	newuser3	0.0		Sept. 22, 2020, 7:45 p.m.	False
4	Psychology in Cinema 20	newuser	0.6153846153846154		Sept. 22, 2020, 7:50 p.m.	False
4	Psychology in Cinema 21	newuser2	1.0000000000000002		Sept. 22, 2020, 7:50 p.m.	False
4	Psychology in Cinema 37	newuser3	0.0		Sept. 22, 2020, 7:50 p.m.	False
4	Psychology in Cinema 20	newuser	0.0		Sept. 22, 2020, 7:55 p.m.	False
4	Psychology in Cinema 21	newuser2	0.0		Sept. 22, 2020, 7:55 p.m.	False
4	Psychology in Cinema 37	newuser3	0.0		Sept. 22, 2020, 7:55 p.m.	False

Figure 6.20: Construct assessment results report – 15 minutes later

Source: Own conception

We can also see that construct assessment results are exported every ten minutes, the results of the construct assessment become available in the Cassandra database (Figure 6.21). The exported results will then be marked in the report of construct assessment results as 'True' in the 'Exported' column.

construct_result_export Geben Sie einen SQL-Ausdruck ein, um die Ergebnisse zu filtern (verwenden Sie Strg+ Leertaste).

	uuid	123 assessment_id	123 course_id	time_created	123 user_id	123 value
1	A3BCC879-5AF2-478F-9576-28A79B42D79F	1.066	4	2020-09-22 17:35:00	21	0
2	EE429E1D-1D75-4AE0-A51B-B5BE0F3F1CD5	1.066	4	2020-09-22 17:35:00	20	0
3	1B0FD226-A942-44C5-94FC-6441AA277887	1.066	4	2020-09-22 17:35:00	37	0
4	30019264-42E9-44C5-9F12-49478C8DA439	1.067	4	2020-09-22 17:40:01	21	1
5	72637D26-2784-417B-9F2D-605A3C79B478	1.067	4	2020-09-22 17:40:01	37	0
6	69E901AF-FAAB-4644-9515-FF9D8CC3E822	1.067	4	2020-09-22 17:40:01	20	0,615
7	33A3AA0B-92F4-4B2F-AE4A-C83A881B6E61	1.068	4	2020-09-22 17:45:01	20	0,615
8	53D824FD-BE15-4CE2-BC7E-16DA77D2B036	1.068	4	2020-09-22 17:45:01	21	1
9	F9948601-8F01-4B0D-A83F-7F6E0170DD1B	1.068	4	2020-09-22 17:45:01	37	0
10	5568A74C-F8DC-4F07-851A-D3FBB48C642B	1.069	4	2020-09-22 17:50:01	20	0,615
11	9490F377-8407-408B-8ACC-C27412559EFC	1.069	4	2020-09-22 17:50:01	21	1
12	C631269E-2FDC-4CF0-963E-DCDC24C88879	1.069	4	2020-09-22 17:50:01	37	0
13	5BA40B81-CF59-4957-A7A5-BC8D834E2821	1.070	4	2020-09-22 17:55:00	21	0
14	B8025B55-5D5E-4731-AB8E-5C72BB113725	1.070	4	2020-09-22 17:55:00	20	0
15	EDA1F15D-62A7-49C3-B892-CF862E058BD1	1.070	4	2020-09-22 17:55:00	37	0

Figure 6.21: Construct assessment results in Cassandra database

Source: as presented in DBeaver application

After a cleanup has taken place, the user can no longer see the exported results in the report of construct assessment results.³

6.3 Review and limitations

The first part of the evaluation showed that the implemented solution satisfies the acceptance criteria of all features and surpasses some of them. Three characteristics of the solution exceed the requirements: The option to individually schedule the assessment of each indicator and construct, the provision of indicator values together with the construct results and the provision of the results as REST service in JSON format.

The second part of the evaluation demonstrated that the solution can be used as desired to measure the construct of Learning Engagement and to schedule assessments for this specific construct. The solution was proven to produce the desired output and to provide the results as expected. Nevertheless, one limitation is that due to time constraints, only one indicator (referring to the indicator 'Forum access' used in Aluja-Banet et al. (2019)) was implemented in order to measure the construct. Therefore, not the whole complexity of the solution could be evaluated here. The solution is however able to measure constructs by a multitude of indicators, which to implement will be object to future work in the case of the Learning Engagement construct.

³ The crontabs of the import of the courses and users, the export of the results of indicator calculations and construct assessments and the cleanup of exported results can be configured in the Django Admin Panel at <http://localhost:8000/api/> after creation of an admin account with the `createsuperuser` management command:
`docker exec -it assessment_ms python manage.py createsuperuser`

7 Discussion

This chapter presents the conclusion of this study. It also presents an outlook to future work.

7.1 Conclusion

In this thesis, the objective was to provide a highly flexible solution for Moodle that performs the assessment of freely configurable constructs and additionally supports the scheduling of construct assessments. This objective has been achieved. The implemented solution has the desired features and performs scheduled construct assessments as expected. Moreover, it allows to assign weights to the indicators for construct assessments and to configure the time range to consider retrospectively for each indicator and each construct. Assessment needs to be activated for courses and users and thereby prevents the unnecessary production of data.

Regarding the research questions, the thesis answered the first question (How can flexibly configurable constructs be assessed using Moodle's log-file data?) by establishing an ORM model of both indicators and constructs and allowing the user to flexibly create instances of these, including the code to execute for the indicators. This allows for an iterative evolution of the measurement approach for any construct.

The second research question (How can construct assessments be scheduled in order to obtain the results regularly?) was answered by applying parametrized celery tasks calling the `save_result` methods of the aforementioned ORM indicator and construct instances, and by setting up instance-specific celery crontab schedules and periodic tasks together for each and every newly created indicator and construct instance. The use of the `django-celery-beat` extension provides the additional benefit of storing these schedules and tasks in the database.

Finally, the third research question (How can interoperability be achieved for the results of the construct assessments?) was answered by implementing a REST API that provides the results of construct assessment and their underlying indicator values as nested relationships, enriched with metadata. This API offers the results in JSON format, a universal format that many external applications can process. Additionally, historic assessment results are persisted in a Cassandra database.

The implemented solution provides a high degree of flexibility in three aspects: First, it allows to virtually measure every possible construct by using a flexible number of indicators that each can access the whole Moodle database. Second, it offers the whole power of the `scikit-learn` preprocessing package to the user, containing several advanced transformer classes to scale the indicator values before they are aggregated. Third, the user can preview the assessment results as well as individually schedule each single indicator and construct. This provides the option to study the response of indicators to certain events in the data and thus iteratively improve the measurement of constructs over time.

On the other hand, the implemented solution has a few limitations itself. First, the opportunity to 'code' indicators by the user might bear a security issue if the solution is not run in a protected network. A possible approach to overcome this limitation would be to parse the indicator code before its execution and only to execute certain whitelisted commands. Second, learners can not configure how their data are supposed to be used. This limitation grounds on the typical unrestricted database access to Moodle by the admin users. Third, the weights of all indicators of one construct might not sum up to one. No good solution for this problem was found, it was decided to leave it to the user to make sure the weights are correctly set.

Eventually, the solution was exemplarily demonstrated to perform scheduled assessments of the construct of Learning Engagement. It was shown that the solution produces the desired output (engagement values) and provides the results in an appropriate way. Specifically, the results of the assessments are provided in a web report and the latest results are made available through a REST API in JSON format. Historical results are persisted in a NoSQL database.

The exemplary implementation of scheduled construct assessments for the construct of Learning Engagement had one limitation though: Out of time constraints, the measurement of the construct in the evaluation chapter was based on only one indicator, which can not be considered a comprehensive measurement of the construct. It thus lacks other indicators considering the multiple facets of the construct. Nevertheless, the purpose of the evaluation was to prove the functioning of the implemented mechanism for scheduled construct assessments.

As a variety of previously used indicators were derived from the existing literature in section 3.2, this thesis already prepared the ground for further improvements of the measurement of the construct of Learning Engagement. As described in section 2.1, the assessment results could be used to identify students at risk, to improve teaching or to support self-regulated learning.

7.2 Outlook

There is a long list of topics for future work. Regarding the user interface, a validation of the indicator code would enhance the user experience. The user would most likely also benefit from an improved navigation inside the app, maybe by adding a start page that offers a dashboard-like overview about constructs and indicators. In general, the design could be enhanced by providing a CSS stylesheet as many objects of the web frontend are functional but not styled yet.

When it comes to the results, it seems a good idea to extend the use of the JSON format also to the export of the historic results to the NoSQL database. The JSON serialization was implemented only late in the project, so this could not be achieved anymore. Nevertheless, it could increase the interoperability of the historic results as well.

Considering the scheduling functionality, it would be a great advancement for the resilience of the implemented solution if the celery worker and the celery-beat scheduler would be daemonized so they don't have to be started separately. This could also not be realized anymore because of time constraints.

Finally, Django's user authentication module could be used to protect the web frontend including the REST API from unauthorized access. This would become relevant when the solution is used in a non-private or insecure network.

List of Tables

3.1	Tables for data collection from Moodle database	12
3.2	Behavioral indicators of motivation by group	15
3.3	Possible weekly engagement scores	18
3.4	Discussion participation metrics	19
4.1	Features from section 4.2 and their acceptance criteria	24
6.1	Acceptance criteria for 'Pending assessments identification' (4.2.1)	47
6.2	Acceptance criteria for 'Assessment scheduling' (4.2.2)	48
6.3	Acceptance criteria for 'Assessment performance' (4.2.3)	48
6.4	Acceptance criteria for 'Results storage' (4.2.4)	49
6.5	Acceptance criteria for 'Results report' (4.2.5)	49
6.6	Acceptance criteria for 'Results emission' (4.2.6)	50

List of Figures

2.1	Learning Engagement detection methods	4
3.1	Process of xAPI statement formation and emission	11
5.1	Sample indicator code	35
5.2	Draft JSON response	44
6.1	Courses listed after import	52
6.2	Confirmation page for course activation	52
6.3	Users listed after import	53
6.4	Empty indicator list	54
6.5	Empty indicator creation form	54
6.6	Filled indicator creation form	55
6.7	Indicator list with one indicator	56
6.8	Indicator ad-hoc values report	56
6.9	Empty construct creation form	57
6.10	Filled construct creation form	57
6.11	Construct list with one construct	58
6.12	Indicator list filtered for construct	58
6.13	Indicator weights update list for construct	59
6.14	Transformed indicator values for construct	59
6.15	Construct ad-hoc values report	60
6.16	Construct schedule configuration	60
6.17	Construct assessment activation	61
6.18	Construct assessment results report	61
6.19	Construct assessment latest results in REST API	62
6.20	Construct assessment results report – 15 minutes later	63
6.21	Construct assessment results in Cassandra database	64

Bibliography

- Akçapınar, G., & Bayazit, A. (2019). Moodleminer: Data mining analysis tool for moodle learning management system. *Elementary Education Online*, 18(1), 406–415.
- Aluja-Banet, T., Sancho, M.-R., & Vukic, I. (2019). Measuring motivation from the virtual learning environment in secondary education. *Journal of Computational Science*, 36.
- Black, P., & Wiliam, D. (2010). Inside the Black Box: Raising Standards through Classroom Assessment. *Phi Delta Kappan*, 92(1), 81–90.
- Black, P., & Wiliam, D. (2018). Classroom assessment and pedagogy. *Assessment in Education: Principles, Policy & Practice*, 25(6), 551–575.
- Bosch, N. (2016). Detecting Student Engagement: Human Versus Machine, In *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*, Halifax, Nova Scotia, Canada, Association for Computing Machinery.
- Chaffai, A., Hassouni, L., & Anoun, H. (2017). E-Learning Real Time Analysis Using Large Scale Infrastructure, In *Proceedings of the 2nd International Conference on Big Data, Cloud and Applications*, New York, NY, USA, Association for Computing Machinery.
- Chatti, M. A., Muslim, A., & Schroeder, U. (2016). Toward an Open Learning Analytics Ecosystem. In *Big Data and Learning Analytics in Higher Education: Current Theory and Practice* (195–219).
- Conde, M. A., García, Á. H., Peñalvo, F. J. G., & Echaluze, M. L. S. (2015). Exploring Student Interactions: Learning Analytics Tools for Student Tracking, In *Learning and Collaboration Technologies*.
- Cooper, A. (2014). *Learning Analytics and Interoperability – The Big Picture in Brief*. Retrieved April 25, 2020, from <http://laceproject.eu/publications/briefing-01.pdf>
- Cooper, A. (2015). *Specifications and Standards - Quick Reference Guide*. Retrieved April 25, 2020, from <http://www.laceproject.eu/wp-content/uploads/2014/11/3-Specifications-and-Standards-Quick-Reference-Guide.pdf>
- Dewan, M., Murshed, M., & Lin, F. (2019). Engagement detection in online learning: a review. *Smart Learning Environments*, 6.
- Di Mitri, D., Scheffel, M., Drachsler, H., Börner, D., Ternier, S., & Specht, M. (2017). Learning pulse: a machine learning approach for predicting performance in self-regulated learning using multi-modal data. In *LAK '17: Proceedings of the Seventh International Learning Analytics Knowledge Conference* (188–197).
- DiCerbo, K., Shute, V., & Kim, Y. (2016). The Future of Assessment in Technology-Rich Environments: Psychometric Considerations. In M. Spector, B. B. Lockee, & M. D. Childress (Eds.), *Learning, Design, and Technology. An International Compendium of Theory, Research, Practice, and Policy* (1–21). New York, NY, USA, Springer.
- Dondorf, T., Pyka, C., Gramlich, R., Sewilam, H., & Nacken, H. (2019). LEARNING ANALYTICS SOFTWARE IMPLEMENTATION FOR THE MOODLE LEARNING MANAGEMENT SYSTEM. In *ICERI2019 Proceedings* (6957–6964).

- Drachsler, H., & Kalz, M. (2016). The MOOC and learning analytics innovation cycle (MOLAC): a reflective summary of ongoing research and its challenges. *Journal of Computer Assisted Learning*, 32, 281–290.
- Drăgulescu, B., Bucos, M., & Vasiu, R. (2016). CVLA: Integrating Multiple Analytics Techniques in a Custom Moodle Report. In *Information and Software Technologies. 21st International Conference, ICIST 2015* (115–126).
- European Commission. (2017). *New European Interoperability Framework*. Retrieved April 15, 2020, from https://ec.europa.eu/isa2/sites/isa/files/eif_brochure_final.pdf
- European Union. (2019). *Education and Training MONITOR 2019*. Retrieved May 27, 2020, from <https://ec.europa.eu/education/sites/education/files/document-library-docs/volume-1-2019-education-and-training-monitor.pdf>
- Feild, J., Lewkow, N., Burns, S., & Gebhardt, K. (2018). A Generalized Classifier to Identify Online Learning Tool Disengagement at Scale, In *Proceedings of the 8th International Conference on Learning Analytics and Knowledge*, New York, NY, USA, Association for Computing Machinery.
- Ferguson, R., & Clow, D. (2015). Examining Engagement: Analysing Learner Subpopulations in Massive Open Online Courses (MOOCs), In *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge*, New York, NY, USA, Association for Computing Machinery.
- Fortenbacher, A., Beuste, L., Elkina, M., Kappe, L., Merceron, A., Pursian, A., Schwarzrock, S., & Wenzlaff, B. (2013). LeMo: a Learning Analytics Application Focussing on User Path Analysis and Interactive Visualization. In *2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)* (748–753).
- Fredricks, J. A., Blumenfeld, P. C., & Paris, A. H. (2004). School Engagement: Potential of the Concept, State of the Evidence. *Review of Educational Research*, 74(1), 59–109.
- Greller, W., & Drachsler, H. (2012). Translating Learning into Numbers: A Generic Framework for Learning Analytics. *Educational Technology & Society*, 15(3), 42–57.
- Griffiths, D., & Hoel, T. (2016). *Comparing xAPI and Caliper*. Retrieved April 29, 2020, from http://www.laceproject.eu/wp-content/uploads/2016/09/LACE-review07_xapi-caliper_v1-1.pdf
- HT2 Labs. (2020). *Say Hello to Our Little (Moodle) Plugin...* Retrieved May 19, 2020, from <https://www.ht2labs.com/learning-locker-moodle-plugin-xapi>
- Kevan, J. M., & Ryan, P. R. (2016). Experience api: Flexible, decentralized and activity-centric data collection. *Technology, Knowledge, and Learning*, 21(1), 143–149.
- Kuh, G. D. (2003). What We're Learning About Student Engagement From NSSE: Benchmarks for Effective Educational Practices. *Change: The Magazine of Higher Learning*, 35(2), 24–32.
- Long, P., & Siemens, G. (2011). Penetrating the Fog: Analytics in learning and education. *EDUCAUSE Review*, 46(4), 31–40.
- Looney, J. (2019). *Digital Formative Assessment: A review of the literature*. Retrieved May 26, 2020, from <http://www.eun.org/documents/411753/817341/Assess%5C%40Learning+Literature+Review/be02d527-8c2f-45e3-9f75-2c5cd596261d>
- Luna, J. M., Castro, C., & Romero, C. (2017). Mdm tool: A data mining framework integrated into moodle. *Computer Applications in Engineering Education*, 25(1), 90–102.
- Monllaó Olivé, D., Huynh, D. Q., Reynolds, M., Dougiamas, M., & Wiese, D. (2018). A Supervised Learning framework for Learning Management Systems. In *DATA '18: Proceedings of the First International Conference on Data Science, E-learning and Information Systems* (1–8).

- Pardo, A., & Kloos, C. D. (2011). Stepping out of the Box: Towards Analytics Outside the Learning Management System, In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*.
- Pardos, Z. A., Baker, R. S. J. D., San Pedro, M. O. C. Z., Gowda, S. M., & Gowda, S. M. (2013). Affective States and State Tests: Investigating How Affect throughout the School Year Predicts End of Year Learning Outcomes, In *Proceedings of the Third International Conference on Learning Analytics and Knowledge*, New York, NY, USA, Association for Computing Machinery.
- Rabelo, T., Lama, M., Amorim, R., & Vidal, J. (2015). SmartLAK: A big data architecture for supporting learning analytics services, In *2015 IEEE Frontiers in Education Conference (FIE)*.
- Romero, C., Romero, J. R., & Ventura, S. (2014). A Survey on Pre-Processing Educational Data. In A. Peña-Ayala (Ed.), *Educational Data Mining. Applications and Trends* (29–64). Springer International Publishing.
- Romero, C., & Ventura, S. (2020). Educational data mining and learning analytics: An updated survey. *WIREs Data Mining and Knowledge Discovery*, 10(3), e1355.
- Romero, C., Ventura, S., & García, E. (2008). Data mining in course management systems: Moodle case study and tutorial. *Computers & Education*, 51, 368–384.
- Samuelsen, J., Chen, W., & Wasson, B. (2019). Integrating multiple data sources for learning analytics—review of literature. *Research and Practice in Technology Enhanced Learning*, 14(11).
- Shum, S. B., & Crick, R. D. (2012). Learning Dispositions and Transferable Competencies: Pedagogy, Modelling and Learning Analytics, In *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge*, New York, NY, USA, Association for Computing Machinery.
- Shute, V. J., Leighton, J. P., Jang, E. E., & Chu, M.-W. (2016). Advances in the Science of Assessment. *Educational Assessment*, 21(1), 34–59.
- Slater, S., Joksimović, S., Kovanovic, V., Baker, R. S., & Gasevic, D. (2017). Tools for Educational Data Mining: A Review. *Journal of Educational and Behavioral Statistics*, 42(1), 85–106.
- Van Lancker, W., & Parolin, Z. (2020). COVID-19, school closures, and child poverty: a social crisis in the making. *The Lancet Public Health*, 5.
- Vidal, J., Rabelo, T., Lama, M., & Amorim, R. (2018). Ontology-based approach for the validation and conformance testing of xAPI events. *Knowledge-Based Systems*, 155, 22–34.
- Wise, A., Zhao, Y., & Hausknecht, S. (2014). Learning Analytics for Online Discussions: Embedded and Extracted Approaches. *Journal of Learning Analytics*, 1, 48–71.
- Yang, D., Lavonen, J. M., & Niemi, H. (2018). Online learning engagement: Critical factors and research evidence from literature.

Appendices

Listing 1: File docker-compose.yml (starting build of the assessment_ms image)

```

1 version: "2.3"
2
3 services:
4   assessment_ms:
5     build:
6       context: ./assessment_ms
7     image: assessment_ms:1.0
8     container_name: assessment_ms
9     depends_on:
10      db:
11        condition: service_healthy
12      cassandra:
13        condition: service_healthy
14     ports:
15       - "${ASSESSMENT_MS_HOST_PORT}:8000"
16     volumes:
17       - "${ASSESSMENT_MS_SOURCE}:/app"
18       - ../.env:/env/.env
19     command: >
20       sh -c "python manage.py runserver 0.0.0.0:8000"
21     restart: unless-stopped
22
23   rabbitmq:
24     image: "rabbitmq:3-management"
25     container_name: rabbitmq
26     hostname: "rabbit"
27     ports:
28       - "${RABBITMQ_HOST_HTTP_API_PORT}:15672"
29       - "${RABBITMQ_HOST_TCP_PORT}:5672"
30     labels:
31       NAME: "rabbitmq"
32     volumes:
33       - "${RABBITMQ_STORAGE}:/var/lib/rabbitmq/mnesia"
34       - "${RABBITMQ_CONFIG}:/etc/rabbitmq/rabbitmq.config"
35     restart: unless-stopped
36
37   moodle:
38     image: 'bitnami/moodle:3'
39     container_name: moodle
40     environment:
41       - "APACHE_HTTP_PORT_NUMBER=${APACHE_HTTP_PORT_NUMBER}"
42       - "APACHE_HTTPS_PORT_NUMBER=${APACHE_HTTPS_PORT_NUMBER}"
43       - "MOODLE_DATABASE_TYPE=${MOODLE_DATABASE_TYPE}"
44       - "MOODLE_DATABASE_HOST=${MOODLE_DATABASE_HOST}"
45       - "MOODLE_DATABASE_PORT_NUMBER=${MOODLE_DATABASE_PORT_NUMBER}"
46       - "MOODLE_DATABASE_NAME=${MOODLE_DATABASE_NAME}"
47       - "MOODLE_DATABASE_USER=${MOODLE_DATABASE_USER}"
48       - "MOODLE_DATABASE_PASSWORD=${MOODLE_DATABASE_PASSWORD}"
49       - "MOODLE_EMAIL=${MOODLE_EMAIL}"
50     ports:
51       - "${MOODLE_HOST_HTTP_PORT}:${APACHE_HTTP_PORT_NUMBER}"
52       - "${MOODLE_HOST_HTTPS_PORT}:${APACHE_HTTPS_PORT_NUMBER}"
53     volumes:
54       - "${MOODLEDATA_PERSISTENCE}:/bitnami"
55     depends_on:

```

```

56     db:
57         condition: service_healthy
58     restart: unless-stopped
59
60 db:
61     image: mysql:5
62     container_name: mysql
63     volumes:
64         - "${MYSQL_PERSISTENCE}:/var/lib/mysql"
65     command: >
66         --character-set-server=utf8mb4
67         --collation-server=utf8mb4_bin
68         --innodb_file_format=barracuda
69         --innodb_file_per_table=On
70         --innodb_large_prefix=On
71
72     environment:
73         - "MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}"
74         - "MYSQL_USER=${MYSQL_USER}"
75         - "MYSQL_PASSWORD=${MYSQL_PASSWORD}"
76         - "MYSQL_DATABASE=${MYSQL_DATABASE}"
77
78     ports:
79         - "${MYSQL_HOST_PORT}:3306"
80
81     healthcheck:
82         test: mysqladmin ping -h 127.0.0.1 -u $$MYSQL_USER --password=$$MYSQL_PASSWORD
83         timeout: 10s
84         retries: 10
85     restart: unless-stopped
86
87 cassandra:
88     image: cassandra:2.1
89     container_name: cassandra
90     ports:
91         - "${CASSANDRA_HOST_TCP_PORT}:9042"
92     volumes:
93         - ./storage/cassandra:/var/lib/cassandra
94     healthcheck:
95         test: ["CMD", "cqlsh", "-u cassandra", "-p cassandra", "-e describe keyspaces"]
96         interval: 15s
97         timeout: 10s
98         retries: 10
99     restart: unless-stopped

```


Listing 2: File .env used by both Docker and the Django project

```

1 #django app env
2 DJANGO_DEBUG=True
3 #DEBUG should be False in production
4 DJANGO_MOODLE_DB_ENGINE=mysql.connector.django
5 DJANGO_MOODLE_DB_NAME=moodle
6 DJANGO_MOODLE_DB_HOST=db
7 DJANGO_MOODLE_DB_USER=moodle
8 DJANGO_MOODLE_DB_PASSWORD=m@0dl3ing
9 DJANGO_MOODLE_DB_RAISE_ON_WARNINGS=True
10 #user and password should be changed in order to have separate user
11 DJANGO_EXPORT_DB_ENGINE=django_cassandra_engine
12 DJANGO_EXPORT_DB_NAME=assessment
13 DJANGO_EXPORT_DB_HOST=cassandra
14 DJANGO_EXPORT_DB_REPLICATION_STRATEGY_CLASS=SimpleStrategy
15 DJANGO_EXPORT_DB_REPLICATION_FACTOR=1
16 DJANGO_EXPORT_DB_CONNECTION_RETRY_CONNECT=True
17 DJANGO_EXPORT_DB_SESSION_DEFAULT_TIMEOUT=10
18 DJANGO_EXPORT_DB_SESSION_DEFAULT_FETCH_SIZE=10000
19 DJANGO_ASSESSMENT_TIME_LIMIT=30
20 DJANGO_APP_LOG_LEVEL=INFO
21 CELERY_LOG_LEVEL=INFO
22 THIRD_PARTY_LOG_LEVEL=INFO
23
24 #assessment_ms env
25 ASSESSMENT_MS_HOST_PORT=127.0.0.1:8000
26 ASSESSMENT_MS_SOURCE=./assessment_ms/app
27
28 #rabbitmq env
29 RABBITMQ_STORAGE=./storage/rabbitmq
30 RABBITMQ_CONFIG=./storage/rabbitmq-config/rabbitmq-isolated.conf
31 RABBITMQ_HOST_HTTP_API_PORT=127.0.0.1:15672
32 RABBITMQ_HOST_TCP_PORT=127.0.0.1:5672
33
34 #moodle env
35 MOODLE_HOST_HTTP_PORT=127.0.0.1:10080
36 MOODLE_HOST_HTTPS_PORT=127.0.0.1:10443
37 MOODLEDATA_PERSISTANCE=./storage/moodledata
38 APACHE_HTTP_PORT_NUMBER=80
39 APACHE_HTTPS_PORT_NUMBER=443
40 MOODLE_DATABASE_TYPE=mysql
41 MOODLE_DATABASE_HOST=db
42 MOODLE_DATABASE_PORT_NUMBER=3306
43 MOODLE_DATABASE_NAME=moodle
44 MOODLE_DATABASE_USER=moodle
45 MOODLE_DATABASE_PASSWORD=m@0dl3ing
46 MOODLE_EMAIL=example@admin.com
47
48 #mysql env
49 MYSQL_PERSISTANCE=./storage/mysql
50 MYSQL_HOST_PORT=127.0.0.1:3306
51 MYSQL_ROOT_PASSWORD=m@0dl3ing
52 MYSQL_USER=moodle
53 MYSQL_PASSWORD=m@0dl3ing
54 MYSQL_DATABASE=moodle
55

```

```

56 #cassandra env
57 CASSANDRA_HOST_TCP_PORT=127.0.0.1:9042

```

Listing 3: Dockerfile of the assessment_ms image

```

1 FROM python:3.7-slim
2
3 RUN pip install --upgrade pip
4
5 ENV PYTHONUNBUFFERED 1
6
7 COPY requirements.txt /requirements.txt
8 RUN pip install -r requirements.txt
9
10 RUN mkdir /app
11 WORKDIR /app
12 COPY ./app /app

```

Listing 4: File requirements.txt of the assessment_ms image

```

1 numpy==1.16.3
2 pandas==0.24.2
3 python-dateutil==2.8.0
4 pytz==2019.1
5 six==1.12.0
6 Django>=2.2.13,<2.3.0
7 mysql-connector-python>=8.0.20,<8.1.0
8 django_ace>=1.0.7,<1.1.0
9 celery >=4.4.7, <4.5.0
10 django-celery-beat
11 cassandra-driver==3.13.0
12 django-cassandra-engine==1.4.0
13 django-environ>=0.4.5,<0.5
14 django-rest-framework==3.11.0
15 scikit-learn==0.23.2

```