An Overview of Moodle's Architecture

Authors:
Ellie Sekine
Brooke Clouston
Ryan Fernandes
Faranak Sharifi-Babaki
Danielle Mott

Abstract

Moodle is a learning management system designed to allow users to create and partake in an interactive educational experience. We explored Moodle by examining the different components of Moodle's system architecture and the style of Moodle's architecture. We looked at Moodle's use of a transaction script and domain model approach, its options for database and filestore implementation, and its support of additional plugins. For each sector of Moodle's architecture, we identified the functional requirements, non-functional requirements, and relevant quality attributes. The organization of these components and how they affect Moodle's stakeholders, mainly content creators and students, are also analyzed. How Moodle is able to ensure a high degree of customizability by allowing administrators to modify internal Moodle features and leverage external resources such as PHP plugins are examined. An overview of Moodle's requirements and an exploration of system scenarios is explained in relation to stakeholders. Moodle's important viewpoints and perspectives are reviewed, along with a functional view of the top-level system and two subsystems; security and evolution. Based on our analysis of Moodle, we provided a proposed extension of the Moodle system to increase Moodle's security

features. We aimed to provide a high level overview of Moodle as a learning management system by studying its architecture.

1) Introduction

In order to understand a software's architecture, one must first understand the motivations behind the creation of the software, the important qualities of the software, the different groups of stakeholders, and the functional and nonfunctional requirements of the software. Once these core pillars are identified, one can begin to fully understand the magnitude of the system. This will allow one to develop a comprehensive overview of the scope of the software and an understanding of how elements of the software's architecture interact to produce a system.

Moodle was created in 2002 by Martin Dougiamas to serve as a learning management system (LMS) [12]. Moodle has since grown to be one of the largest open-source projects with over 200,000 users. The integrity of Moodle is maintained by a team of only 50 developers. The majority of Moodle's development can be attributed to the community of open source developers that contribute to the improvement of Moodle.

Part of what makes Moodle an interesting architecture to study is that it's an open-source web application [13]. Moodle's plugins are a new concept that few other architectures are using. Moodle's philosophy allows all "learners" to contribute to the educational experience. It's used primarily by educational institutions such as elementary schools and universities. However, it's also used in corporate settings to provide training to employees.

In order to give a comprehensive overview of the Moodle software architecture, this report will be broken into 4 sections: system overview, stakeholders and requirements, architectural views and proposed extension of the architecture.

The system overview section will explain the purpose, scope, and audience of Moodle. This section will also identify the software quality attributes that are relevant to Moodle's software and why they are important. Furthermore, this section will describe the system architecture and the architecture style used by Moodle.

Stakeholders and requirements will identify the different groups of stakeholders that are relevant to Moodle and why the stakeholders care about Moodle. This section will also describe some of the most important functional and nonfunctional requirements for Moodle's software. Furthermore, three functional scenarios and three quality scenarios are created in this section.

Architectural views will define the scope of Moodle and how its different elements work together to create the system. An overview of the different viewpoints and perspectives of Moodle will also be discussed. The functional view for both the top-level system and the chosen subsystem with their perspectives are also discussed in this section.

Finally, the proposed extension of the architecture section will define a feature that could be implemented in Moodle, explained with a use case diagram and a sequence diagram.

Understanding a specific software's architecture is the first step to comprehending what the software can and cannot do. Once a developer is aware of a software's features and shortcomings, they are better prepared to improve the existing software and develop superior software in the future.

2) System Overview 2a) Purpose and Scope

Business Goals and Drivers: Moodle is an open-source LMS used to create online courses. Online courses consist of pages that contain resources and activities that are available to students. This is used as a way to facilitate blended learning, flipped classrooms, and distance studies. Some of its core features include a grade book, calendar, themes, notifications, 50 different enrolment verification options, and the ability to add plugins[12]. It is written in PHP in a non-object-oriented style and lets users create their own websites through a GUI, without the need for any coding experience[13]. Users can enhance the features of their websites beyond Moodle's core functionality by installing plugins and using various APIs[12].

Architectural Constraints: Moodle must adhere to various E-Learning standards. It uses the Sharable Content Object Reference Model (SCORM) 1.2, which is a collection of E-Learning specifications that define communication between the client and server sides[12]. In addition, Moodle 2.1 adopted the Aviation Industry Computer-Based Training Committee (AICC) HACP standard. Moodle also acts as a Learning Tools Interoperability (LTI) provider by using a plugin[12]. LTI is a way of integrating learning applications with education.

Code Organization: Moodle code is organized in a transaction script approach that supplies data to the HTML generated by renderer classes[13]. Moodle also uses the domain model by having core functionalities contained in libraries[13]. The Moodle database is an aggregate of core tables and plugin tables[13].

2b) Audience

Stakeholders[14]

Teachers	Those who rely on Moodle to help facilitate learning. Teachers create and provide the resources and activities that students will access through Moodle.
Organizations	Schools, universities, airlines, military organizations, oil companies, and healthcare organizations can use Moodle as a way to facilitate proper training and education for students and employees.
Students	Those who rely on Moodle for learning. They access the resources and activities provided on the site.
Internal Developers	Those who create and maintain Moodle's core functionalities and are employed by Moodle.

Plugin Developers	Anyone in the public that chooses to create APIs in order to extend Moodle's functionality.
Moodle Partners	Experts in creating Moodle sites and courses. They provide training, technical help, and implementation services to those building Moodle sites. They support Moodle through royalties.
Moodle course creators	Those who use Moodle's GUI to create course sites. This is typically a master teacher, department head, or program coordinator.

Table 1.

2c) Why the System is Architecturally Interesting

The three most interesting features of Moodle's architecture are its extensive ability to support plugins, its use of a transaction script and domain model approach, and its database abstraction layer.

Upon installation of Moodle, the web application's libraries and included plugins are sufficient to allow the user to begin using Moodle as an LMS. Yet, Moodle developers recognized each user's needs were unique and designed the system architecture to support the addition and removal of diverse plugins. Different plugins use separate APIs to interact with core libraries, but Moodle ensures that all plugins support its core functionality. The ability to easily add plugins makes Moodle highly customizable [24].

An additional characteristic of Moodle is its use of both a transaction script and a domain model approach. A transaction script allows a group of actions, typically related to interacting with the database, to be performed following the "all-or-nothing" paradigm [7]. It's the logical choice for applications written in PHP, like Moodle's plugins, where each transaction has its own script. Yet it's not the logical choice for a system as large-scale and complex as Moodle. Consequently, Moodle developers reorganized code that executes common functionality into libraries following the domain model approach. This allowed the reuse and organization of code[24].

Another key feature of Moodle's architecture is its use of ADOdb to provide a database abstraction layer. As a result, Moodle is able to support a variety of relational databases including MySQL, PostgreSQL, and Microsoft SQL Server [23]. Developers can manage data, without worrying about the trivial details of the underlying database.

2d) Quality Properties

In this section, interoperability, modifiability, security, and usability will be identified and discussed as Moodle's quality properties.

Moodle's interoperability allows its developers and users to leverage resources outside of those provided by Moodle. This can reduce the time it takes to develop features, increase reliability, and reduce the maintenance required. For example, Moodle's user authentication utilizes the

software protocol LDAP[21]. Moodle developers were not required to write their own implementation of a user authentication protocol, nor are they required to maintain it. Additionally, its users can import external resources, such as quiz questions. This facilitates the user's experience building and using the LMS.
Moodle is an extremely modifiable system. Following an open source workflow, developers can share their changes on Moodle Tracker, have their changes peer reviewed, and ask the developers managing the Moodle repository to examine their changes [20]. Moodle developers will test and possibly approve the changes. The open source workflow enables all users to discover and fix bugs, as well as improve the system. The increased number of developers as compared to a closed source workflow can lead to higher quality code in the repository. Additionally, as previously mentioned, Moodle is designed to support plugins. Plugins allow the user to modify the system to fit their unique needs.

Moodle implements security by assigning each user a role. A role specifies the permissions that a user possesses. It's imperative to the system behavior that a user may not modify their own permissions. An obvious example is that a student should never be able to assign themselves grades. There are a set of default roles including teacher, student, and administrator but it is also possible to create custom roles. Additionally, it's possible to override a role, switch roles, or modify what a user in a specific role may view. This enables Moodle administrators to adapt Moodle security options if the default permissions for a given role don't fit the exact requirements of a user's situation [22].

Usability is an important Moodle quality. Moodle developers ensure usability by maintaining a strong focus on its users, namely educators and students. Moodle conducts usability testing by having someone adept with Moodle analyze typical users as they attempt to complete different Moodle tasks. From usability tests, it can be seen where users struggle. Developers can then improve those areas of Moodle. Additionally, Moodle developers strive to minimize the number of preferences available to a user. This reduces the number of decisions a user can make when executing a task. Usability reduces the time it takes a user to complete a task, increases the ease of use for a user, and ultimately results in Moodle users not switching to a different, easier to use LMS[11, 15].

2e) System Architecture

Figure 1 represents a dependency graph of the functional scenario where the software validates a user on login before they access any course. The top level system represented by this diagram is the subsystems in the classes folder of lib [3]. These subsystems: access, privacy, event, message, task, session, user, and authentication were chosen because they're responsible for validating a legitimate user upon successful entry of corresponding login credentials. The way they're responsible is described in the figure. This was determined by using the *Understand* Tool, where subsystems butterfly graphs were examined to see which other subsystems they depended on. The subsystems included in the diagram are the ones that had the greatest number of file dependencies. Moreover, they were examined in Github, where the files that they included were read to obtain a better understanding of what each subsystem does [3]. The access subsystem was examined in more detail, as it had many interesting functions regarding how it

validated a user's credentials. It had functions like require_login(), is_guest(), is_viewing() and is_enrolled() [1]. It also used authentication plugins, which would allow the user to access the appropriate part of the site for their role if the "auth_user_login" function returned true [2]. The functions that the authentication plugins would call would also make use of other plugins, such as privacy, user related, standard login and admin tools plugins, which relate to other subsystems [2].

Subsystem	**What it does**
Message	Moodle messaging API will send messages to Moodle users. Messages could include, "This account is invalid," etc.
Task	The task subsystem deals with any operations that takes more than a few seconds. Files in task that related to logging in are send_failed_login_notifications.php and session_cleanup_task.php.
Session	Session manager is the public Moodle API for sessions. Manages when sessions start and end.
User	An informal ground of miscellaneous user-related APIs relating to sorting and searching lists of users.
Access	The access API gives you functions so you can determine what the current user is allowed to do. It includes functions like require_login(), isloggedin().
Authentication	The authentication API describes Moddle's interface functions to authentication plugins.
Event	When an action takes place, an event is created by a core API or plugin. The Event subsystem includes files such as user_loggedin.php, user_loggedinas.php.
Privacy	Provides users with more control over their data and how it is processed.

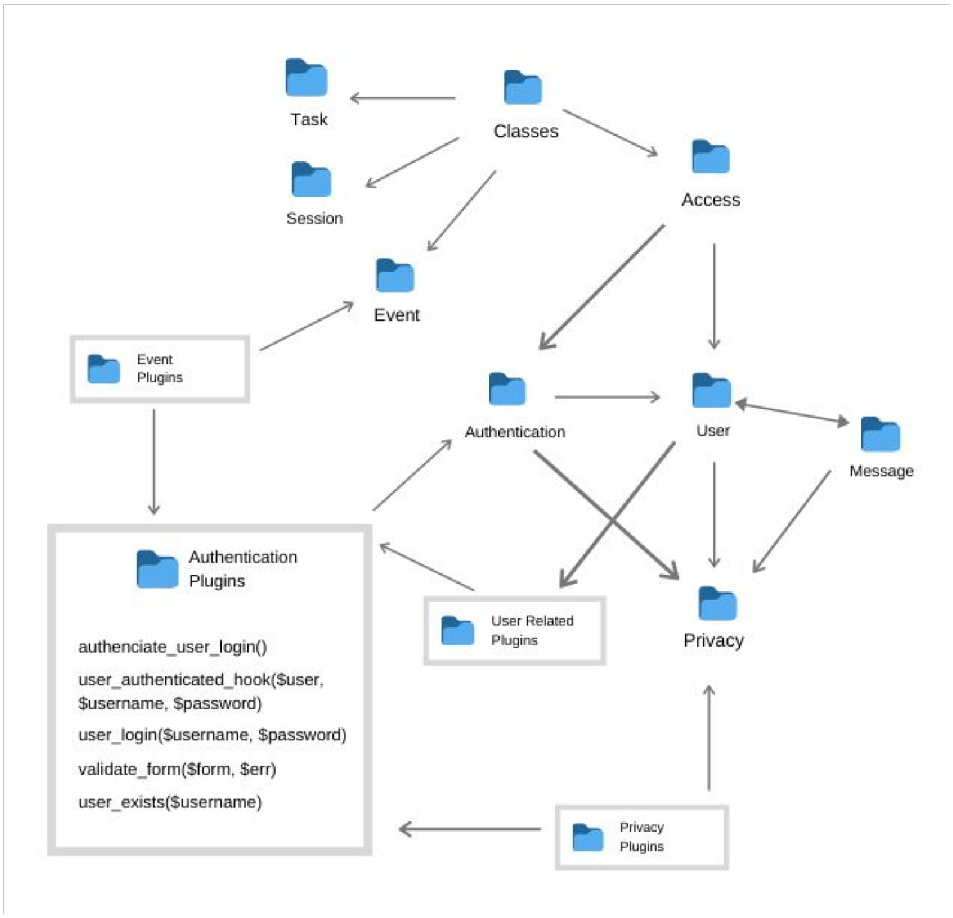


Figure 1. **2f) Architectural Style**

The architecture implements a plug-in architecture style[9]. The plug-in style contains a core system, the foundational logic, and plug-ins, the additional features that extend the core[10]. Moodle has an extensive core that provides the basic functionalities to an LMS, including course enrollment, course navigation, and user functionalities. Moodle has 35 different types of plug-ins. It's also strongly typed, so depending on what type of function you want the plug-in to perform, you may have to write a different API. Plug-ins in Moodle are structured as a folder of files that are named using the type as the prefix and the name of the specific plug-in as the postfix.

3) Stakeholders and Requirements 3a) Stakeholders

Throughout its existence, Moodle has gathered a number of stakeholders . The different types of stakeholders for the Moodle system range from educational clients to other organizations that incorporate Moodle as an e-learning experience. Since Moodle is an open-source project, it can be inspected, modified and enhanced by the Moodle community.

Since Moodle's development, it has acquired a vast number of stakeholders that can be separated into several categories. Table 2 mentions the most relevant stakeholder categories.

Stakeholder Class	**Description: Why they care about it**

Acquirers Moodle Partners, Moodle Users Association and Moodle HQ [4]	Oversee the procurement of the system or product. Moodle Partners provides critical funding to support the open source project.[4] Moodle HQ distributed the system under the GNU General Public License. Members from the Moodle Users Association also help fund the Moodle projects.
Communicators Moodle HQ and Volunteers	Explain the system to other stakeholders via its documentation and training materials. Moodle HQ provides the documentation for understanding how to get started with Moodle. Volunteers work on translating Module into 140 languages in order to make documentation accessible to everyone.
Developers Martin Dougiamas and Moodle HQ. Moodle community can be considered as the key developers of Moodle.	Construct and deploy the system from specifications (or lead the team that do this). Martin Dougiamas, the original creator of Moodle, developed Moodle to help educators

	build online courses that focus on collaboration. Moodle HQ is the core team involved in the development process with approximately 50 developers. The Moodle community contributes to the Moodle project and works on creating plugins and themes.

Maintainers Moodle HQ is the core team responsible for maintaining Moodle. Clients working with Moodle Partners also help with maintenance.	Manage the evolution of the system once it is operational. Moodle HQ forms the roadmap for the development process. Clients working with Moodle Partners are able to provide feedback and have their proposed features included in the system development. The amount paid by the client to Moodle Partners is used to fund the open source project. [4]
Support Staff Moodle Partners, certified by Moodle HQ.	Provide support to users for the product or system when it is running Moodle Partners provide training, support and any other Moodle help that the client needs, for a price. [4]
System Administrator MoodleHQ and Administrator on Moodle site	Run the system once it has been deployed. The system is run by the core team MoodleHQ.Within the Moodle site, the user given the Administrator role, can delegate tasks, edit features, and assign manager roles to a user.
Testers MoodleHQ and Moodle community developers	Test the system to ensure that it is suitable for use. MoodleHQ tests the new features before adding it to the critical system. Moodle community developers test their code before submitting a pull request into the main depository. Testing is needed to check if the code does what it's supposed to do.

Users Managers, teachers, students, and parents can all be considered end users in the educational sector of the Moodle system. Privacy officers in organisations with GDPR compliance and other organisations. Open source community/Moodle Community.	Define the system's functionality and ultimately make use of it. Managers have the ability to build, manage and supervise how Moodle features are portrayed in their respective websites. They can also customize the site layout. Managers are also given the ability to moderate any courses and categories, as well as manage the roles of the system users. Teachers use Moodle to enroll students, record feedback, and manage and set up the rubric for student grades. Students use Moodle to complete their assignments and collaborate with their peers. Most of the student's evaluations and deliveries for their courses take place on Moodle. This makes it an important system for students. The role of parent has the least number of features in Moodle. As users, they're allowed to view grades and other information without accessing their child's courses. They can also monitor their child's activity and progress in courses. Moodle provides a role for privacy officers to organizations with GDPR compliance (regulations specified in EU laws on data protection). Privacy officers can view data requests and respond to queries. In the workplace, Moodle is used to deliver effective corporate training and professional development. With Moodle being a free and open-source LMS, it's used by casual and professional developers alike. This community of developers collaborates in the expansion of Moodle.

Table 2.

3b) Overview of Requirements

Functional and non-functional requirements are an integral part of any system architecture, especially large scale architectures with many users such as Moodle. A functional requirement defines the components of a system. It essentially describes what functions a software must perform. Functional requirements are mandatory and help to verify the functionality of the software. A non-functional requirement defines the qualities of a system and helps determine how well a specific part of the system is executed. Non-functional requirements are non-mandatory, but they're essential to verify the performance of the software[8].
Some examples of functional requirements in Moodles architecture are displayed in Table 3.1.

Functional requirement	**Description**
The software shall validate a user on login.	Moodle must check a user's login data against a database to ensure the user trying to log in is a valid-user.
The software shall determine the user type on login.	A user who is a student must be able to log in and see the courses they are enrolled in, while a user who is an administrator (i.e. a teacher) must be able to log in and see the course they are teaching.
The software shall allow for actions that are specific to the user type.	A user who is a student must be allowed to perform student actions such as view lecture notes, take quizzes and participate in online discussions. A user who is a course administrator must be allowed to perform administrative actions such as being able to update marks, create quizzes and publish lecture notes. There should be no overlap in the actions specific users can do.
The software shall implement the user-specific theme.	A user from university X that is accessing the software must be presented with university X's theme on login. Themes should be dependent on the type of user that is accessing the server.

The software shall be secure.	Passwords and user data should be encrypted so that there are no data leaks which could impact a student and/or course administrator .
The software shall be extensible for multiple	Moodle shall be able to accommodate
course learning operations and assessment operations	different learning materials such as text files, video files, powerpoints, discussion forums, etc. Moodle should also incorporate options for multiple-choice quizzes, written answer quizzes, etc.
Moodle shall be available in multiple different languages .	Moodle shall provide language support in order to increase client availability.
The software shall be open source.	Moodle software shall be open-source to allow for community-driven feedback and development.
The software shall be able to recover from failures quickly and accordingly.	Moodle software should have built-in failure recognition that allows for graceful catching of errors. For example, failure recognition is triggered when a user is accessing a page that doesn't exist or trying to perform a task they're not authorized to do.
The software shall be easily modifiable.	Moodle software should be written in a way such that developers are able to easily add additional features (or plugins) that are extensible.
The software shall run quickly and efficiently.	Moodle software should be able to load, accept, process, execute, and respond to user commands quickly and efficiently.

Table 3.1.

Some examples of Moodle's non-functional architecture features are provided in Table 3.2:

Non-functional requirement	**Description**

Students shall never be allowed to update their grades.	User roles must have restrictions on what actions they can and cannot perform.
Moodle shall be able to handle many users with various roles who are accessing the site at once.	Moodle must be able to work efficiently and maintain speed performance when it is under “heavy stress” caused by many users accessing it at once.
The software shall be portable so that moving from one OS to another OS does not create	The type of OS a user is using should not interfere with the quality of performance they receive from Moodle.

any problems.	
Moodle shall have a clear and concise user interface for all types of users.	The operations a user is able to perform should be clear and laid out in an intuitive way. This enables a user with little or no e-learning experience to quickly grasp how to use Moodle.
Moodle shall be well documented to allow for potential contributors to get up to speed quickly.	By having comprehensive and well-documented source code, a developer looking to contribute to the Moodle infrastructure will quickly be able to be brought up to speed.
Existing Moodle code and newly created plugins shall be easily testable.	Designing code that is easily testable allows for a quicker and smoother integration into the existing codebase. It also ensures no major outages will occur.
Moodle source code shall have a high degree of readability.	Source code shall adhere to proper guidelines set in place by the Moodle development team to ensure a high degree of understanding for others reading it.

Moodle shall be reliable.	Moodle shall be reliable so that students and course administrators are constantly able to access the platform and learn effectively from the platform.
Moodle shall have a high degree of data integrity.	Moodle shall implement a high degree of integrity when it comes to storing sensitive user data such as emails, grades, and personal information.
Moodle shall be maintained.	All source code shall be written in a way that allows for easy maintenance.
Moodle shall be efficient.	Moodle shall require a database server for clients to distribute content to their users. Too large or too complex of a set up may hinder client acceptance.
Moodle shall be robust.	Moodle shall be able to perform in the presence of faults, stress, invalid inputs, etc.

Table 3.2.

3c) System Scenarios Functional Scenarios

Overview: How the system validates a legitimate user upon successful entry of corresponding login credentials.

System State: Assuming that the user is registered with Moodle, there's an entry in an external database which contains the user's corresponding login ID and password.

System Overview: The section of the system responsible for validating a user is waiting idly for a user to enter their login credentials and submit their credentials for verification. The rest of the system is running properly.

External Stimulus: The user enters their login credentials and submits them for verification.

Required System Response: The system calls the Authentication API with the user's username and login [5]. If the "auth_user_login" function returns true, the user is allowed to access the appropriate part of the site for their role.

Overview: How the system ensures a user of a specific type is allowed to perform an action.

System State: There exists a multidimensional array called "accessdata" which contains

information regarding the role assignments, any role switches, and relevant time entries [30].

Each user's related data is stored in this array.

System Environment: The system is running properly and the user is logged into their Moodle account.

External Stimulus: The user attempts to perform an action.

Required System Response: Before allowing the user to perform the requested action, the system must check that the user is in a role that is allowed to perform this action. The system calls the function "has_capability" in the Access API (accesslib.php) with the necessary parameters [5]. If this function returns the boolean true, the user is allowed to perform the action, otherwise the user won't be allowed to perform the action.

Overview: How the system responds to a manager or teacher specifying a language for their course.

System State: There are language pack(s) installed by a Moodle site administrator [29]. There is a default language set for the site, but this may have been overridden in different places [28]. **System Environment:** The system is running properly and a site administrator is logged in. **External Stimulus:** A manager or teacher requests to change the language of the course by going to Course Administration, Edit, Settings, and then Force language for the desired language[28].

Required System Response: The system will apply the selected language pack (found in the lang directory) to the course content so that the course is displayed in the language of choice [29].

Quality Scenarios

Overview: How the system uses multiple web servers to handle large amounts of network traffic.

System State: There is a database, filestore, and cache areas for the specific implementation of the Moodle site[27].

System Environment: The system is using multiple web servers to handle the multitude of requests. The different web servers access the same database, filestore, and cache areas. They're consequently able to handle the same types of requests[27, 28].

External Stimulus: The Moodle website is receiving hundreds, thousands, or millions (depending on the Moodle site) of requests at the same time from different users in various roles. The site must respond to each of the user's requests.

Required System Response: The load balancer will direct user requests to different capable web servers. This will allow the site to process requests as fast possible and prevent a single server from exceeding capacity [27, 28].

Overview: How the system implements planned outages.
System State: The "auth_outage" plugin is installed and enabled.
System Environment: The system is running a version of Moodle 3.3 or higher [26].
External Stimulus: A user in an administrative role or with necessary permissions arranges the planned outage by navigating to the site Dashboard, then Site administration, then Plugins, then Authentication, then Outage manager, and finally the Manage tab. At the correct time intervals, the system will respond according to what the user scheduled [26].
Required System Response: At a fixed time before the outage occurs, a warning forecasting the planned outage will be displayed on the website. When the outage occurs, there will be a different warning bar describing the implications of the outage and the time required for the outage. Post-outage, there will be a notification communicating that the system is up and running again. Note how the outage administrator configured the outage will affect how the events of the outage unfolded [26].

Overview: How the system is able to be safely upgraded.
System State: The Moodle software, uploaded files, and database have all been backed up. The old Moodle software program files have been removed from the server and stored somewhere else. The upgrade file is unzipped where the out-dated software program files used to be and the config.php file is moved back to the new directory. All up-to-date plugins installed on to the site have been added to the new Moodle directory. The moodledata folder and code are not stored in the same folder.
System Environment: The system is using a server that meets the requirements for Moodle 3.7, the site is in maintenance mode, and all cron processes are stopped. All plugins are up to date. **External Stimulus:** Maintenance mode is stopped and the update is initiated by the site administrator.
Required System Response: The system will discover the new version of the system. The corresponding database and/or filestore will perform the specified upgrades. When the system upgrades successfully, there will be no error messages and the system will be ready for use by all users.

4) Architectural Views 4a) Context View

Moodle is a web application licensed under the GNU General Public License. It's written partially by open source developers and managed by Moodle Perth Australia. It's mainly coded in PHP, but also includes HTML, CSS, and JavaScript. It can be extended by adding plugins. Some available Moodle plugins are authentication, enrollment, and repository plugins. Moodle also has numerous APIs, which provide tools for Moodle scripts. Some of the more commonly used APIs allow data access, data manipulation, file organization, forms, logging in, and navigation.

Moodle can be run on Unix, Linux, FreeBSD, Windows, OS X, and Netware. It can be installed on a PHP-capable web server. It also includes a database managed by MySQL, PostgreSQL, Microsoft SQL Server, MariaDB, or Oracle. Additionally, there's a file store for uploaded and generated files.

Moodle is used by different organizations in the e-learning sector, including schools and companies. It supports e-learning standards such as SCORM, AICC HACP, and IMS Content Cartridge Packages. Moodle competes with BlackBoard, Desire2Learn, Instructure, and Canvas. Although BlackBoard became Moodle's official partner in March 2016, this partnership was terminated later. In Europe, Latin America and Oceania, Moodle has over 50% of the market share as of May 2019. However, in the American higher education market, Moodle has only 18% of the market share as of fall 2018.

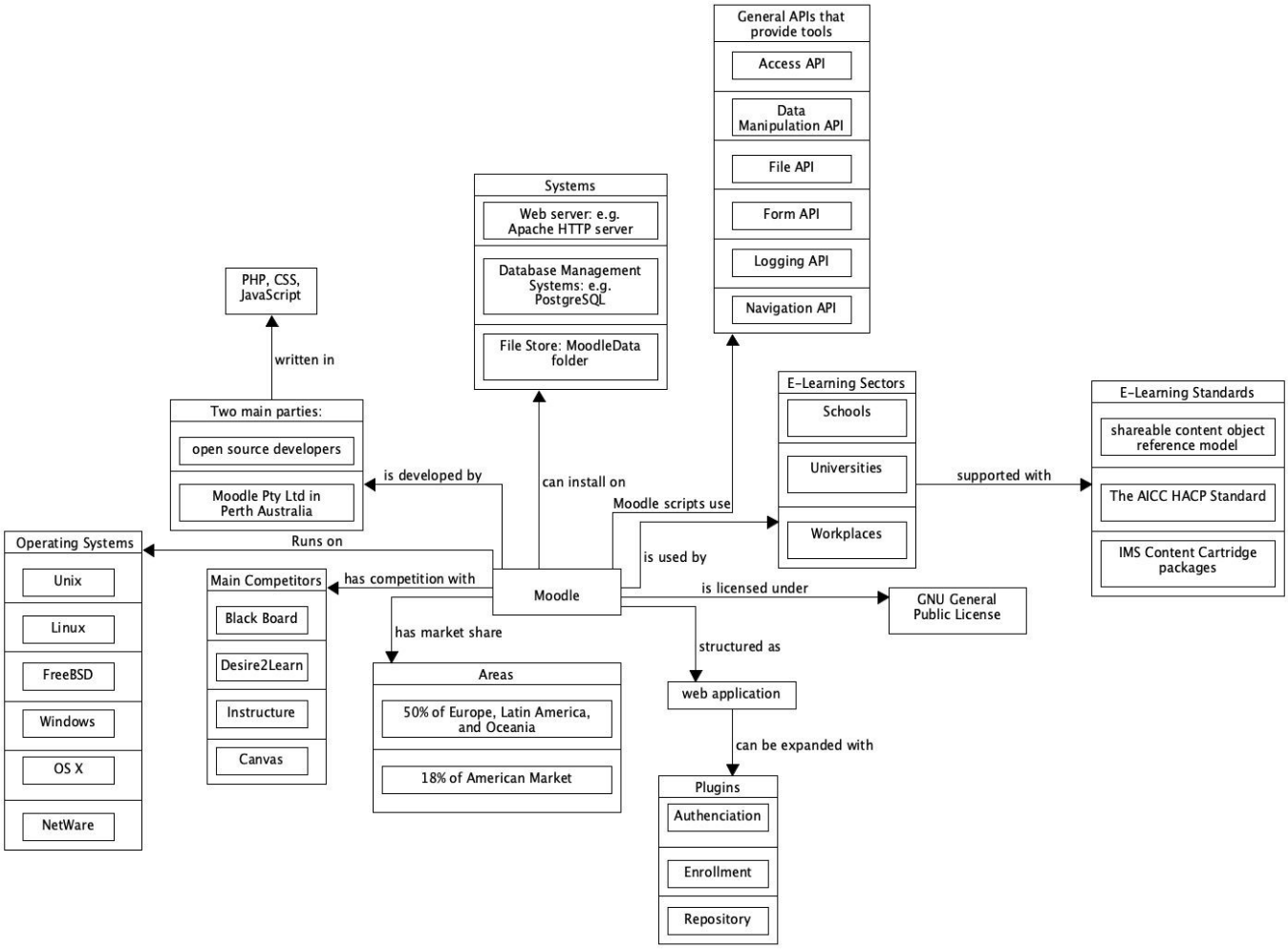


Figure 2.

4b) Overview of Viewpoints and Perspectives

High: *Functional, Development, Scalability, Deployment, Tool Support, Performance, Supportability, Development Efficiency, Modifiability*

Viewpoints that are central to Moodle include the functional, development and deployment viewpoints. It's essential that the software elements from different sections of the Moodle source code interact properly with one another. Furthermore, it's important that the architecture is able to support the building, testing and maintenance of the system. This means the software and network dependencies of Moodle must be well integrated. These viewpoints have an effect on the development efficiency and modifiability of Moodle. One must also consider the perspective of scalability because Moodle is used by many users all over the globe.

Additionally, Moodle uses many APIs that can provide tools for Moodle scripts. Moodle's APIs must be considered when studying the system. Finally, it's important to Moodle's end users, such as students and teachers, that the system performs well, meets performance demands, and is easy to use. It's also necessary that Moodle is supported on a variety of devices.

Medium: *Information, Concurrency, Operational, Availability, Reliability, Portability, Time to deliver.*

Moodle stores and distributes lots of information. Concurrent processes such as learning activities should run well on the system. Thus information and concurrency viewpoints are principal to Moodle. Moreover, it's necessary that how Moodle is used, administered and supported while running in production is communicated to its many developers. Thus the operational viewpoint becomes central. Also, Moodle has many users and developers worldwide. Consequently, Moodle must be available, reliable, portable and able to meet the performance demands for all users and stakeholders.

Low: *Security, Resilience, Safety*
Even though it's necessary for Moodle to be safe and secure, security isn't that high of a priority. Since much of the information stored via Moodle relates to student academic performance, this data is not considered sensitive. Thus perspectives such as security, resilience, and safety are not of high importance.

4c) Functional View of the Top Level System
Functional viewpoints describe the system's functional elements, their responsibilities, interfaces, and primary interactions[16]. The following table lists some functional viewpoints and describes them through perspectives that are important to Moodle at a top-level system.

Functional Elements of Moodle at a Top-Level	**Description**
Accessibility	Different applications and plugins have different options, such as audio, video and text display formats that can aid people with disabilities.
Performance	Specific features about Moodle which help to increase the performance of Moodle, such as Moodle's database abstraction level which reduces load on the client side.
Development Resources	Moodle's ability to be designed, built, deployed, and operated within known constraints around people, budget, time and materials. This is done through an open-source approach.

Evolution	Moodle's ability to be flexible in the face of inevitable changes that systems experience after deployment, balanced against the costs of providing such flexibility. Moodle must be able to identify a problem and its developers must implement a change quickly when a
	problem is identified.
Location of elements	The ability of Moodle and its developers to design elements in such a way that overcome problems brought about by the absolute location of its elements and the distances between them.
Security	Moodle's ability to reliably control, monitor, and audit who can perform what actions on what resources. Moodle must also be able to detect and recover from any failures in security mechanisms such as security breaches or attacks.
Usability	The ease with which people who interact with Moodle can work efficiently and effectively. This includes developers who are contributing to Moodle's architecture as well as users who are trying to access Moodle's features.

Table 4.1.

The functional viewpoints accessibility and evolution will be more thoroughly examined through viewpoints as they relate to Moodle.

Two important perspective related to Moodle as a Top Level System:

Perspective	Description

Accessibility	From a top-level view, it's critical that Moodle shall be accessible to anyone who wants to use it. Due to the nature of Moodle, an online learning tool, it must be able to accommodate people who have different disabilities, including but not limited to: people who have vision disabilities, people who have hearing disabilities, and people who have learning disabilities. Some of the approaches that Moodle takes to address these accessibility perspectives include having videos with audio (for users who have vision disabilities and may not be able to see a lecture slide), videos with subtitles and/or text documents (for users who have hearing impairment and may not be able to hear audio on a video), and support for multiple presentation methods for those who have learning disabilities (videos, powerpoints, quizzes, etc.). By accommodating for a variety of user's accessibility needs, Moodle ensures all of its users are able to learn effectively and in a way that suits them best.
Evolution	From a top-level view, Moodle shall be able to evolve efficiently. Moodle addresses the topic of evolution by allowing its users to become part of the development team and have a say in what its new features are. This unique approach is made possible due to Moodle's open-source nature. By allowing its users to be involved in the development process, Moodle ensures that it is constantly evolving to conform to what the users (and often stakeholders) want. This approach ensures that Moodle is never outdated or out of touch with the wants and needs of its end users and stakeholders. This approach ensures its end users are satisfied with the product they are paying for and allows them to continue using Moodle in the way that suits them best.

Table 4.2.

Top-Level Sequence Diagram:

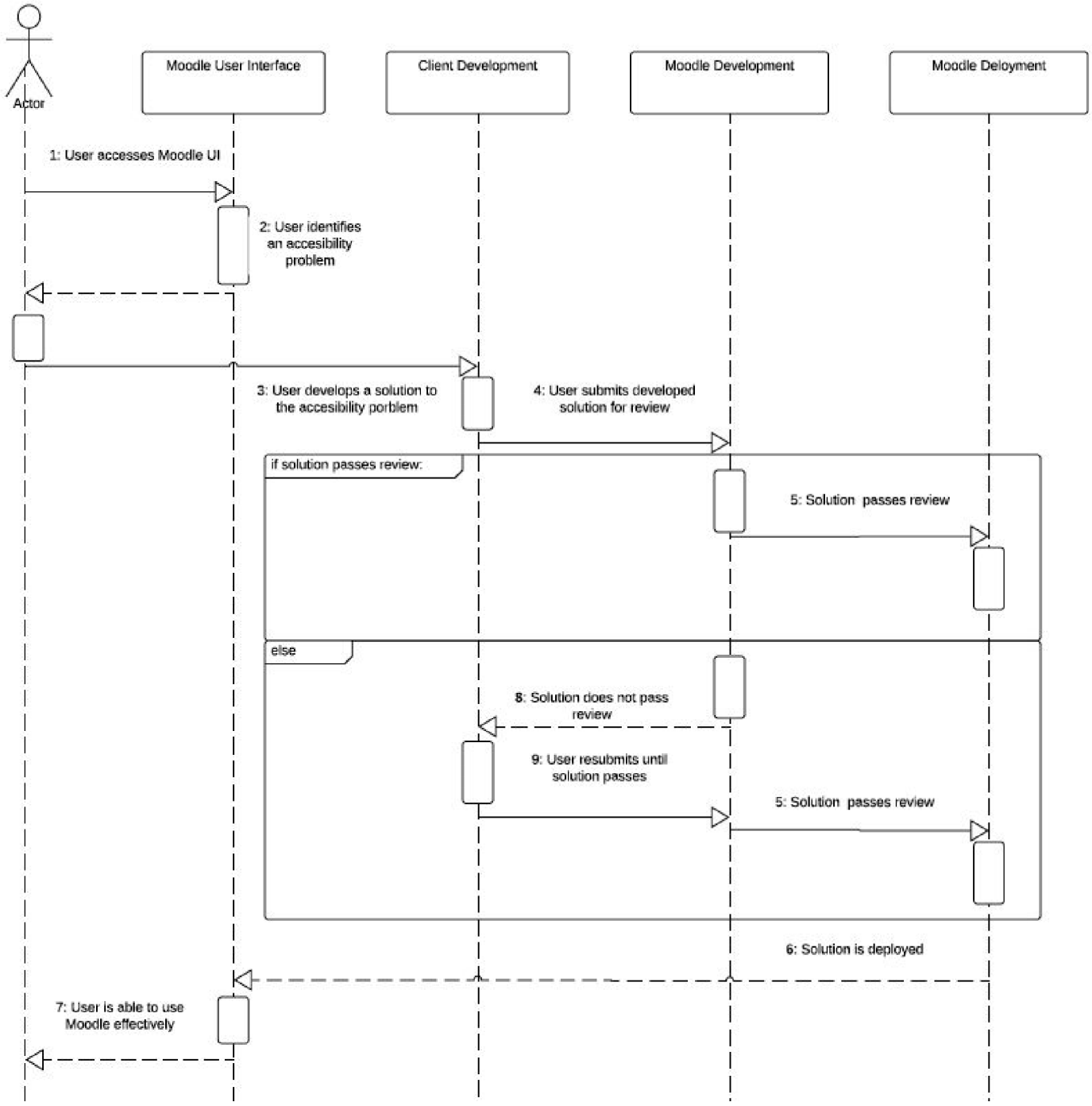


Figure 3.

Top-Level Sequence Diagram description:

Sequence Action	Description
1. User Accesses Moodle UI	The user logs into their Moodle account from the user interface and is met with their personalized settings.
2. User Identifies an accessibility problem	The user discovers an accessibility problem when they are trying to access Moodles attributes.
3. User develops a solution to the accessibility problem	The user develops what they believe is a viable solution to the problem that other users may benefit from if they are struggling with the same accessibility problem.
4. User submits the developed solution for review	The user sends their proposed solution and source code to the Moodle development team for review.
5. Solution passes review	The proposed solution passes review.
6. Solution is deployed	The user's proposed solution is deployed to the active Moodle user interface for use by all users.
7. User is able to use Moodle efficiently	The user is able to use Moodle efficiently and have their accessibility needs met.
8. Solution does not pass review	The user's proposed solution does not meet the criteria for Moodle's development team.
9. User resubmits until solution passes	The user makes the necessary changes outlined by the Moodle development team and resubmits their solution. This process continues until the solution is accepted.

Table 5.

4d) Functional View of the Chosen Subsystem: Access Subsystem

Access subsystem provides various functionalities to determine what the current user is allowed to do [5].

Functional View of Access Subsystem	
Functional Element	Access control, user capability, login confirmation [Refer: Table 6.3 contains the elements responsibilities]
Interface	● User enters username and password that is processed by the Authentication API [6]. ○ If inputs are true, the user will have access to the homepage of Moodle. ● Between the Access Control and the User Capability, the has_capability() function is invoked which also takes in the context instance generated from Access Control. It checks if the user has a particular capability with the given context and provides a boolean expression: True, if valid or else it throws an exception. [5] *E.g.* : does the user "student" have the capability to change their grades?
Connector	● Subsystem elements interact with each other based on received boolean values and/or outputs from functions within functional elements. ● Context instances, are instantiated within the Access control element to be used by certain functions like has_capability(), require_capability() and then deleted automatically by the system. [5] ● Communication between elements of Access subsystem are synchronous.
External Entities	Authentication plugins, Authentication API, other top level systems

Table 6.1.

Two important perspective related to Access Subsystem:

Perspective	Description
Security	Moodle incorporates the Authentication subsystem and Access subsystem in ensuring that the user logged in can see only content related to their roles and whether the user has the capability of performing certain tasks, based on its context. In terms of security Access subsystem only ensures that the right user can see the content and perform the activities, it does not provide any additional security feature for the information being stored.

Evolution	Moodle uses the Access subsystem as a core API, using its functions in various plugins. Based on how the API was developed, the Access API allows modules to extend Moodle with new capability. [5] New capabilities can come in the form of new functional requirements that can use Access Subsystem to determine whether the user has the capability or not.

Table 6.2.

Subsystem sequence diagram functional elements and responsibilities:

Functional Elements	**Responsibility**
Authentication API → Core Subsystem of Moodle	● Interacts with the Access subsystem and authenticates user inputs with the external database. ● It describes Moodle's Interface functions to authentication plugins.
Access Control	● Input information is sent from Authentication API to the User Capability and other components within the Access API. ● Context interfaces are created relating to users and used in the User Capability element.
User Capability	● Calls has_capability() function, checks whether a user has a particular capability in a given context. If yes, it will return true. ● If the user does not have the capability, the require_capability() function will throw an access control exception.

Login Confirmation	This element has multiple functions related to the user's initially interaction with Moodle. ● Provides function require_login() after setting up the page ○ This function verifies that the user is logged in before accessing any course or activities ○ Verifies access to hidden courses and activities ○ Logs access to course ● Provides require_course_login() ○ Used only in activities that want to allow read access to content on the frontpage without logging-in.

Table 6.3.

Sequence diagram related to the scenario that describes how the system validates a legitimate user to access any course upon successful entry of corresponding login credentials:

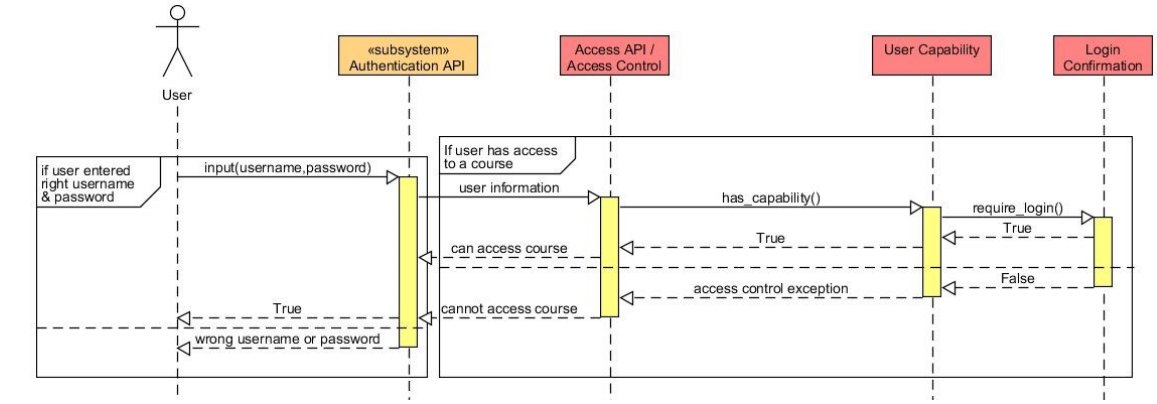


Figure 4.

Sequence Diagram related to the scenario that describes how the system ensures a user of a specific type is allowed to perform an action:

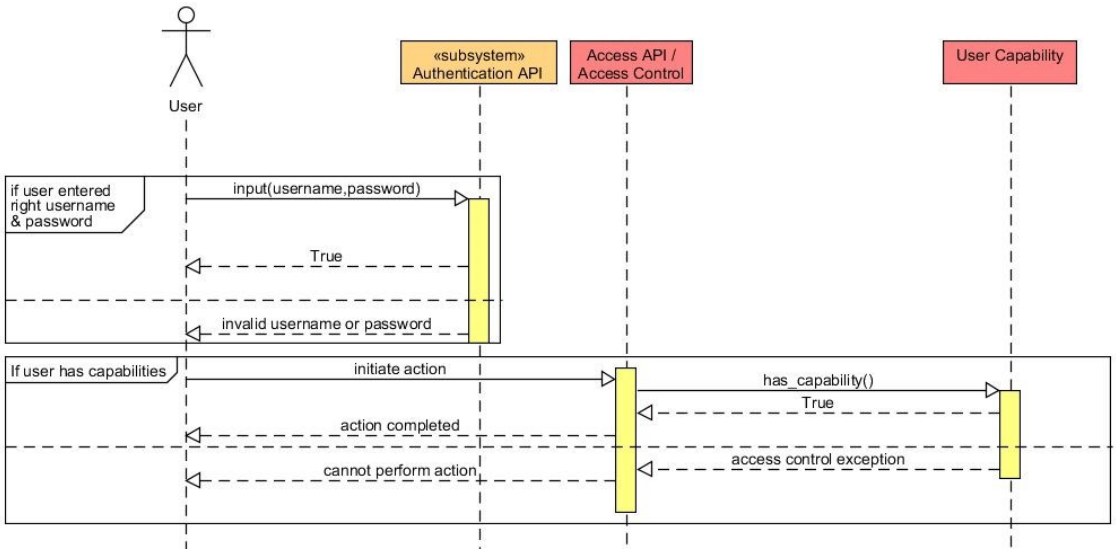


Figure 5.

5) Proposed Extension of the Architecture

Our team's proposed enhancement is to include a timer that would prevent a user from making countless attempts to login into the system. For example, if the user inputs the wrong username and/or password repeatedly five times, the user would temporarily be banned from logging in for fifteen seconds. Any subsequent failed login attempts would result in an increase of the temporary ban. This proposed enhancement could be included within the Authentication subsystem, when the system checks the user's input against the database. The proposed feature would be beneficial from a security perspective. The proposed feature would also enhance Moodle's security and act as an obstacle or a preventive measure for any attackers trying to gain access into the system using the brute force attack method.

Functional Requirements

Moodle shall keep a count of failed login attempts.	There will be an entry in Moodle's database devoted to recording the number of failed logins for a specific user. It updates in real time.
Moodle shall keep track of time in between failed login attempts	Moodle shall keep track of time in between failed login attempts. Furthermore, Moodle shall prohibit a user from attempting to login again for a period of 15 seconds after 5 unsuccessful login attempts.

Quality Requirements

Moodle shall display an error message if login credentials are invalid.	If the user enters an incorrect password for an existing Moodle account, a message will be displayed to the user indicating they entered an invalid password.
Moodle shall display an error message if the user has reached the maximum number of failed attempts to login.	If the user enters an incorrect password for the 5th time, a message will be displayed informing the user that they will not be able to attempt to login for 15 seconds. The message will disappear after the 15 seconds.
Moodle shall be able to access the database to check credentials and be able to return values quickly.	It is important for the communication between the Moodle server and its database to be quick so that the user does not have to wait a long time while their credentials are validated.
Upon the event of incorrect credentials entered, Moodle shall require new text be entered in the text boxes.	This is to ensure new information was actually supplied and the user is not repeatedly getting the same response error.

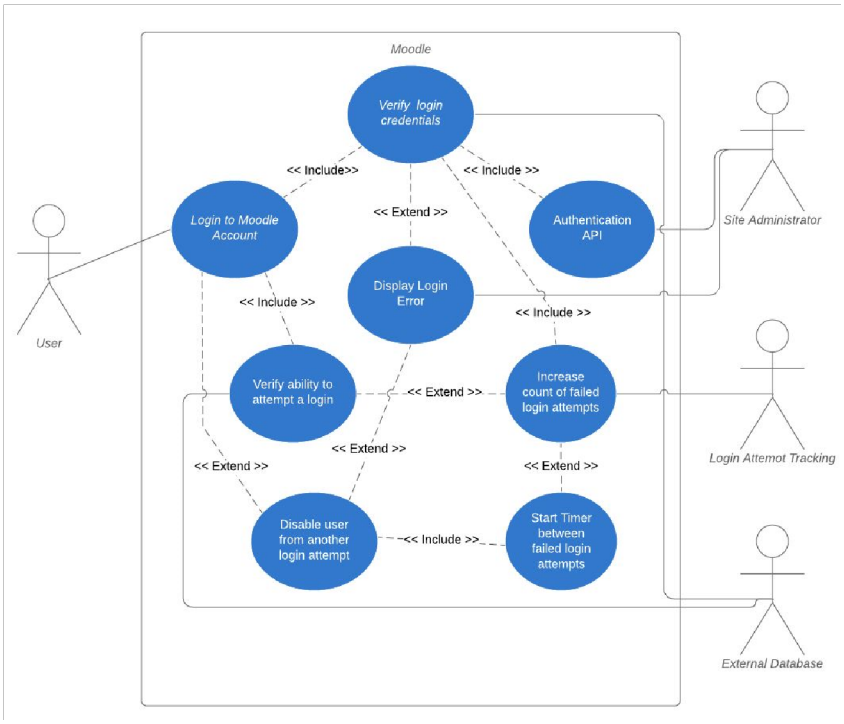
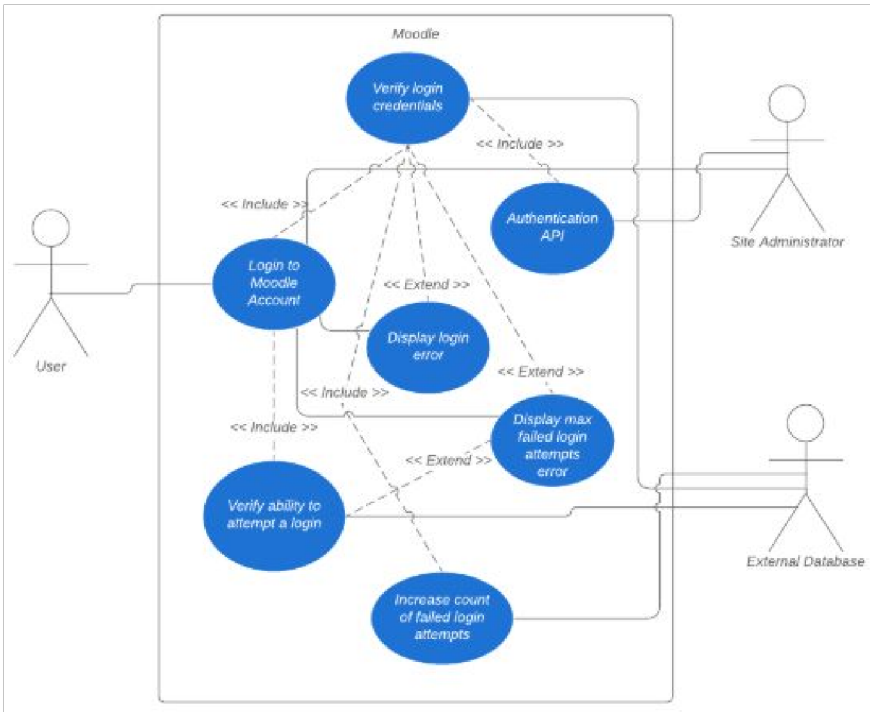


Figure 6.1.

In Figure 6.1, observe a user attempting to log into their Moodle account. If they are successful then they are able to gain access with a full range of their actions. If not, a count of the user's attempts to login is incremented and once the maximum count threshold has been reached a timer is started to prohibit the user from attempting to login while the lockout period is still in effect. The timing functionality of the feature is an important functional requirement for the enhancement because, in order for the system to put restraints on the frequency at which a user is able to attempt to log in, it must be able to keep track of the time passed in between failed login attempts.

Figure 6.2.

In Figure 6.2, we have the use case diagram for a user logging into their account. Upon entry of their login credentials, the system verifies the credentials by calling the Authentication API which in turn queries an external database. If they incorrectly enter their login credentials, the count of incorrect attempts is incremented. If they attempt to login, fail to enter their correct login credentials, and have reached the maximum number of tries, an error message will be displayed indicating that they must wait in order to attempt to login again.



The following table breaks down how different subsystems and interfaces of the current architecture can support the proposed enhancement. The authentication API is the interface that will have to communicate the most with the other subsystems to successfully carry out this enhancement of Moodle's login feature.

Subsystem	How it will be changed for the enhancement
Authentication	The authentication API is responsible for checking the user input with the database. However, in the circumstance that the user has failed to login correctly five times in a row, then on the sixth attempt, the authentication API should communicate with the task subsystem, which would implement a timer. The time should be communicated to the authentication API since it needs to wait and not process any username and password before it can check if the user's input is valid on the subsequent try. The authentication subsystem would also need to communicate with the event subsystem, to keep track of the value of the counter, which reflects how many times a user has failed to login. It would also need to communicate with the message subsystem to send the correct message, either saying that they can try again or that they have to wait. In the case of a brute force attack by a bot, it is suggested that every attempt is stored in a database, by the authentication subsystem. This is for the reason that later, Moodle can go back and note the attacks that have been made against the site.
Message	The message subsystem is responsible for sending messages to Moodle users, and when the user tries to login after five (or a multiple of five) times, the message will need to be changed to something like, "Login attempt failed five times- please wait fifteen seconds."
Event	The event subsystem creates an event when an action takes place. This can include keeping track of the number of times that the user has failed to login. The event subsystem would include a counter, that would tell the user to wait fifteen seconds when counter reaches a value that is a multiple of five. Also, the event subsystem should note the date and time of each attempted login. This can be helpful when identifying a bot, since in that circumstance the time difference of each attempted login would be minimum.
Task	The task subsystem deals with any operation that takes more than a few seconds. Files in task that relate to logging in are send_failed_login_notifications.php. This would need to be changed because on the login after the fifth (or a multiple of five) try, it would also need to implement a timer for fifteen seconds, and send a notification to the authentication subsystem to wait.

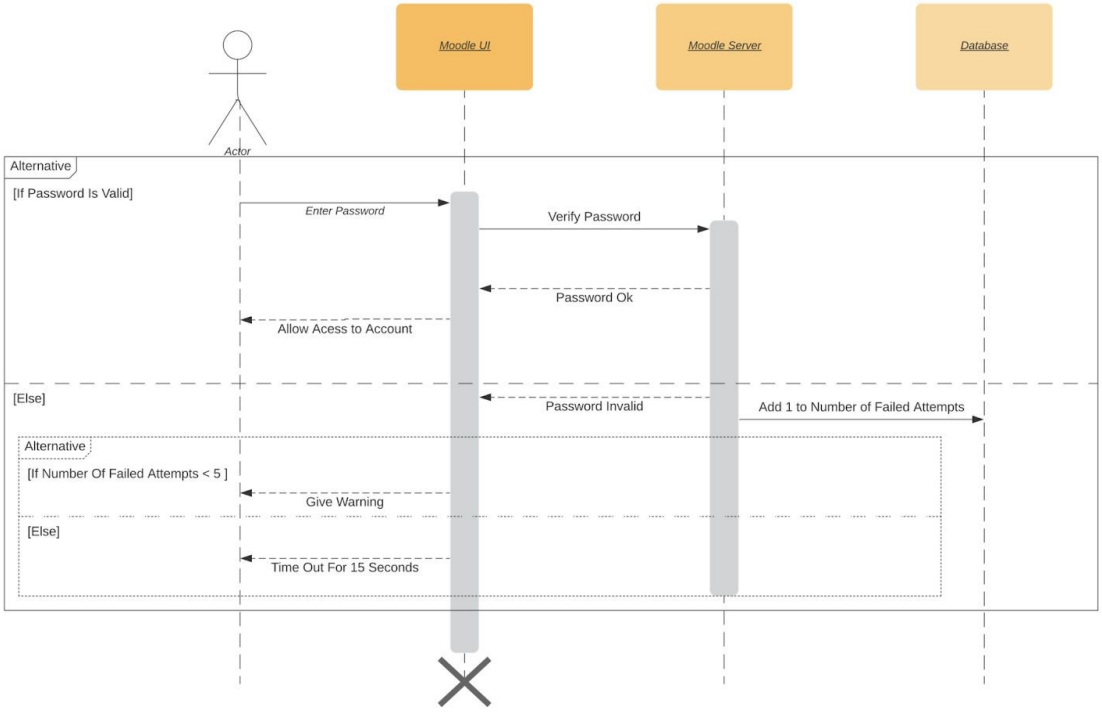


Figure 6.3.

6) Conclusions

Moodle is an LMS that was developed to create online courses as a way to make learning accessible and flexible. Being an application where people without coding experience can create websites and it's extensive ability to add plugins, Moodle is way a to provide a customizable learning experience.

Moodle is a platform that relies on a community of developers. Since Moodle is open source with a plug-in style architecture, external developers can modify and improve the system, and create fully custom plugins in order to extend Moodle's core functionality. It is also highly interoperable, and allows for developers and users to incorporate third party resources.

Moodle uses a transaction script and domain model architecture. Although the transaction script architecture can make sense for applications written in PHP, it’s not a logical choice for one as large scale as Moodle.

From a functional top level view, Moodle is a platform that needs to be very accessible and be one that can easily evolve. Moodle’s access API is vital to its extensibility, and it needs to be evolvable and secure.

We suggest that Moodle adds an enhancement that prevents the user from making countless attempts at logging in by banning the user from attempting log-in for 15 seconds after 5 failed attempts.

In conclusion, with it's high customizability and interoperability, Moodle is a great LMS for any user wanting to create a tailored learning experience. Through the large community of Moodle users and developers, there are many ways Moodle can continue to be improved and for its core functionalities to be extended upon.

6) References

[1] "Core APIs," *Core APIs - MoodleDocs* , 15-Jan-2019. [Online]. Available: https://docs.moodle.org/dev/Core_APIs. [Accessed: 17-Oct-2019].

[2] "Plugin types," *Plugin types - MoodleDocs* , 15-Jan-2019. [Online]. Available: https://docs.moodle.org/dev/Plugin_types. [Accessed: 10-Nov-2019].

[3] "Moodle," *Github* , 23-Sept-2010. [Online]. Available:https://github.com/moodle/moodle. [Accessed: 10-Nov-2019]. [4] "Advantages of Working with a Certified Moodle Partner – eThink," *eThink Education | Hosted Moodle & Totara LMS*, 21-Oct-2019. [Online]. Available: https://ethinkeducation.com/blog/whats-a-certified-moodle-partner-advantages-expert-moodle-host/. [Accessed: 14-Nov-2019].

[5] "Access API," *Access API - MoodleDocs* . [Online]. Available: https://docs.moodle.org/dev/Access_API. [Accessed: 14-Nov-2019].

[6] "Authentication API," *Authentication API - MoodleDocs* . [Online]. Available: https://docs.moodle.org/dev/Authentication_API. [Accessed: 14-Nov-2019]. [7] M. Fowler, "Transaction Script," *P of EAA: Transaction Script*. [Online]. Available: https://martinfowler.com/eaaCatalog/transactionScript.html. [Accessed: 17-Oct-2019].

[8] "Functional Requirements vs Non Functional Requirements: Key Differences," *Guru99*. [Online]. Available: https://www.guru99.com/functional-vs-non-functional-requirements.html. [Accessed: 17-Oct-2019].

[9] T. Hunt, "Moodle," *The Architecture of Open Source Applications (Volume 2): Moodle*. [Online]. Available: http://www.aosabook.org/en/moodle.html. [Accessed: 17-Oct-2019].

[10] F. T. Imam, "Documenting a Software Architecture," *CISC 322/326 Course Notes* , pp. 115–127.

[11] S. Levy, "Understanding Moodle Usability - Moodle Usability Series, Part 1," *Lambda Solutions*. [Online]. Available: https://www.lambdasolutions.net/blog/understanding-moodle-usability-moodle-usability-series-part-1. [Accessed: 17-Oct-2019].

[12] "Moodle," *Wikipedia* , 16-Oct-2019. [Online]. Available: https://en.wikipedia.org/wiki/Moodle. [Accessed: 17-Oct-2019].

[13] "Moodle architecture," *Moodle architecture - MoodleDocs* , 22-Aug-2018. [Online]. Available: https://docs.moodle.org/dev/Moodle_architecture. [Accessed: 17-Oct-2019].

[14] "Usage" *Usage - MoodleDocs* , 07-Oct-2019. [Online]. Available: https://docs.moodle.org/37/en/Usage. [Accessed: 17-Oct-2019].

[15] "Usability," *Usability - MoodleDocs* . [Online]. Available: https://docs.moodle.org/dev/Usability. [Accessed: 14-Nov-2019]. [16] F. T. Imam, "Revist Architectural Views," *CISC 322/326 Course Notes* , pp. 20–22.

[17] "Moodle," *The Architecture of Open Source Applications (Volume 2): Moodle* . [Online]. Available: https://www.aosabook.org/en/moodle.html. [Accessed: 14-Nov-2019].

[18] O. Elgabry, "Plug-in Architecture," *Medium* , 01-May-2019. [Online]. Available: https://medium.com/omarelgabrys-blog/plug-in-architecture-dec207291800. [Accessed: 14-Nov-2019].

[19] "Moodle architecture," *Moodle architecture - MoodleDocs* . [Online]. Available: https://docs.moodle.org/dev/Moodle_architecture#An_overview_of_Moodle_core. [Accessed: 14-Nov-2019].

[20] "Tracker introduction," *Tracker introduction - MoodleDocs* . [Online]. Available: https://docs.moodle.org/dev/Tracker_introduction. [Accessed: 14-Nov-2019]. [21] "Roles and permissions," *Roles and permissions - MoodleDocs* . [Online]. Available: https://docs.moodle.org/37/en/Roles_and_permissions. [Accessed: 14-Nov-2019].

[22] "Standards," *Standards - MoodleDocs* . [Online]. Available: https://docs.moodle.org/37/en/Standards. [Accessed: 14-Nov-2019].

[23] "Database," *Database - MoodleDocs* . [Online]. Available: https://docs.moodle.org/37/en/Database. [Accessed: 14-Nov-2019].

[24] "Moodle architecture," Moodle architecture - MoodleDocs. [Online]. Available: https://docs.moodle.org/dev/Moodle_architecture#Moodle_as_a_modular_system. [Accessed: 14-Nov-2019].

[25] "Upgrading," *Upgrading - MoodleDocs*. [Online]. Available: https://docs.moodle.org/37/en/Upgrading#Check_for_plugin_updates. [Accessed:

14-Nov-2019].

[26] Catalyst, “catalyst/moodle-auth_outage,” *GitHub* , 14-Nov-2019. [Online]. Available: https://github.com/catalyst/moodle-auth_outage#what-is-this. [Accessed: 14-Nov-2019].

[27] “Performance recommendations,” *Performance recommendations - MoodleDocs* . [Online]. Available: https://docs.moodle.org/37/en/Performance_recommendations#Scalability. [Accessed: 14-Nov-2019].

[28] “Language settings,” *Language settings - MoodleDocs* . [Online]. Available: https://docs.moodle.org/37/en/Language_settings. [Accessed: 14-Nov-2019].

[29] “What Is Load Balancing? How Load Balancers Work,” *NGINX* . [Online]. Available: https://www.nginx.com/resources/glossary/load-balancing/. [Accessed: 14-Nov-2019].

[30] “Language packs,” *Language packs - MoodleDocs* . [Online]. Available: https://docs.moodle.org/37/en/Language_packs. [Accessed: 14-Nov-2019].

[31] Moodle, “moodle/moodle,” *GitHub* . [Online]. Available: https://github.com/moodle/moodle/blob/master/lib/accesslib.php. [Accessed: 14-Nov-2019].