António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Part 1 - Review of T32's Library System Architecture

Topic	Library System Architecture
Diagrams	Very Good
Clarity	Very Good: the description was accurate and helpful to better understand all scenarios.
Consistency	Very Good
Completeness	Good: you could've explained what actions the user can have with each block
Sufficient Level of detail	Excellent
Text Description	Very Good
Clarity	Good: sometimes the text could be more descriptive on how the modules work with each other.
Consistency	Very good
Sufficient	Very Good: adequate
Correctness	Very Good
Anything missed?	Very Good: more diagrams would help to further understand the relation between modules and actions
Will it work?	Very Good: It would probably work
Summary	Very Good: more detail on the diagrams would improve the overall understanding of the architecture.

* Qualitative scale: Excellent (19-20), Very good (17-18), Good (14-16), Fair (10-13), Poor (<10)

Explanation

Diagrams

- Clarity: Are all the figures clear? Is it easy to understand what they mean? Are they laid out in an intuitive way? Is it clear what colors mean? If necessary, is there a legend?
- Consistency: Are the shapes and lines used consistently? Is there anything (a shape or line) that means two different things? Are colors consistent, if used?
- Complete: Is anything missing that would make the diagrams clearer?
- Sufficient level of detail: Do the diagrams give enough detail? For example, are there some components that should be broken down more?

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Text Description

- Clarity: How clear is the description?
- Consistency: How consistent is the description with the diagrams?
- Sufficient: Does the text give enough information?

Correctness

- Did the architects miss anything, such as did they forget a component?
- Will anything not work?

Summary

- How confident are you that you could implement the design as it is described?

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Part 2 – Understanding **Moodle**'s Architecture

- ## Summary of the Architecture

Introduction and Background

Moodle is a web application used in educational settings, designed to facilitate online learning and teaching. The document aims to provide a comprehensive overview of Moodle, focusing on its architectural elements, including its plugin-based structure, permission system, output generation, and database abstraction layer. These components collectively contribute to Moodle's effectiveness as a virtual learning environment (VLE) or learning or course management system (LMS, CMS or even LCMS).

This architecture was developed by Martin Dougiamas in 1999 at Curtin University, Australia, and it has evolved significantly from its first release in 2002. Written in PHP, it is compatible with most web servers and supports various database systems, such as MySQL, PostgreSQL, Microsoft SQL Server, and Oracle. The project's inception was driven by the need for a platform that supported social constructivist pedagogies, enabling users to learn collaboratively in an online environment.

Architecture and Design

Plugin-Based Structure

One of Moodle's core design philosophies is its modular architecture, which allows for extensive customization and extension through plugins. This approach facilitates the academic community's ability to adapt the system to specific needs without specialty in the use of the technology. From the document, the Moodle's architecture (ARCH) supports approximately 35 different types of plugins, ranging from activity modules to authentication and question types, each requiring the implementation of a specific API.

Permission System

A sophisticated role and capability-based permission system is another key point of Moodle's ARCH. It allows some control over user actions across different contexts within the system, such as system-wide, course categories, individual courses, and activities within courses, like online exams & material sharing. This flexibility enables administrators to finely tune access permissions under the educational institution's policies and needs, without any special knowledge regarding the ARCH.

Output Generation

Moodle's design allows for the application of different themes/skins and supports localization for various languages, enhancing its global usability. The separation of logic and presentation in its design facilitates the customization of the user interface without modifying the core functionality[1].

1. Author's Note: When reading the chapter "13.4. Generating Output", on the document that this summary is about, we imagined the University of Porto's Moodle, regarding the unique design compared to other Portuguese Universities Moodle.

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Database Abstraction Layer

Moodle employs a database abstraction layer that supports various Relational Database Management Systems (RDBMS), enabling it to operate on different platforms. This layer abstracts the database operations, ensuring that Moodle can function efficiently regardless of the underlying database system.

Implementation Details

Moodle's implementation details revealed its adaptability and scalability. This system comprises three primary components: the codebase, the database, and the *moodledata* folder. These components are designed to work seamlessly, whether on a single server or across a distributed environment, ensuring the system performance and reliability.

Request Dispatching and URL handling

Moodle's approach to handling web requests is straightforward, using standard PHP mechanisms to map URLs to specific scripts. This simplicity in design makes it easier for the developers of it to understand and work with/on the system, despite the potential for perceived inelegance from the user's perspective.

Roles and Permissions System

The intricate roles and permissions system in Moodle allows for detailed access control, configurable at various contextual levels within the system. This system is essential for creating a flexible learning environment where users can have different roles and permissions across different courses and activities[2].

Database Interaction

Moodle's database layer facilitates interaction with the database in a way that it abstracts the complexities of direct SQL queries, offering methods for performing operations that are both secure and efficient. This design choice highlights Moodle's commitment to security and performance, accommodating the diverse needs of its user base.

Development and Community Involvement

The Moodle project benefits significantly from its open-source nature, with a global community of developers contributing to its ongoing development and improvement, typically on the development of plugins. This collaborative model has enabled Moodle to adapt and evolve in response to the changing needs of the educational sector and technological advancements.

Conclusion

Moodle's architecture and design reflect a deep understanding of the needs of the educational community it serves. Its plugin-based structure, comprehensive permission system, flexible output generation capabilities, and robust database abstraction layer collectively provide a powerful and versatile platform for online learning. As Moodle continues to evolve, it remains at the forefront of educational technology, driven by a committed community of developers and users dedicated to encouraging an engaging and effective learning environment.

2. "For example, a user might be a Teacher in one course, and a Student in another, and so have different permissions in each place." (page 6)

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

- # Diagrams of the Architecture

At the beginning of the document, the author provides a summary of Moodle's entities using only text. However, we believe it's beneficial to complement this with an UML representation of the entire architecture. Although Moodle is segmented into plugins, there are core entities within its system that are crucial for understanding Moodle's purpose, functionality, and the relationships between entities. Of course, this UML only represents Moodle itself, not representing external entities that can be integrated in Moodle.

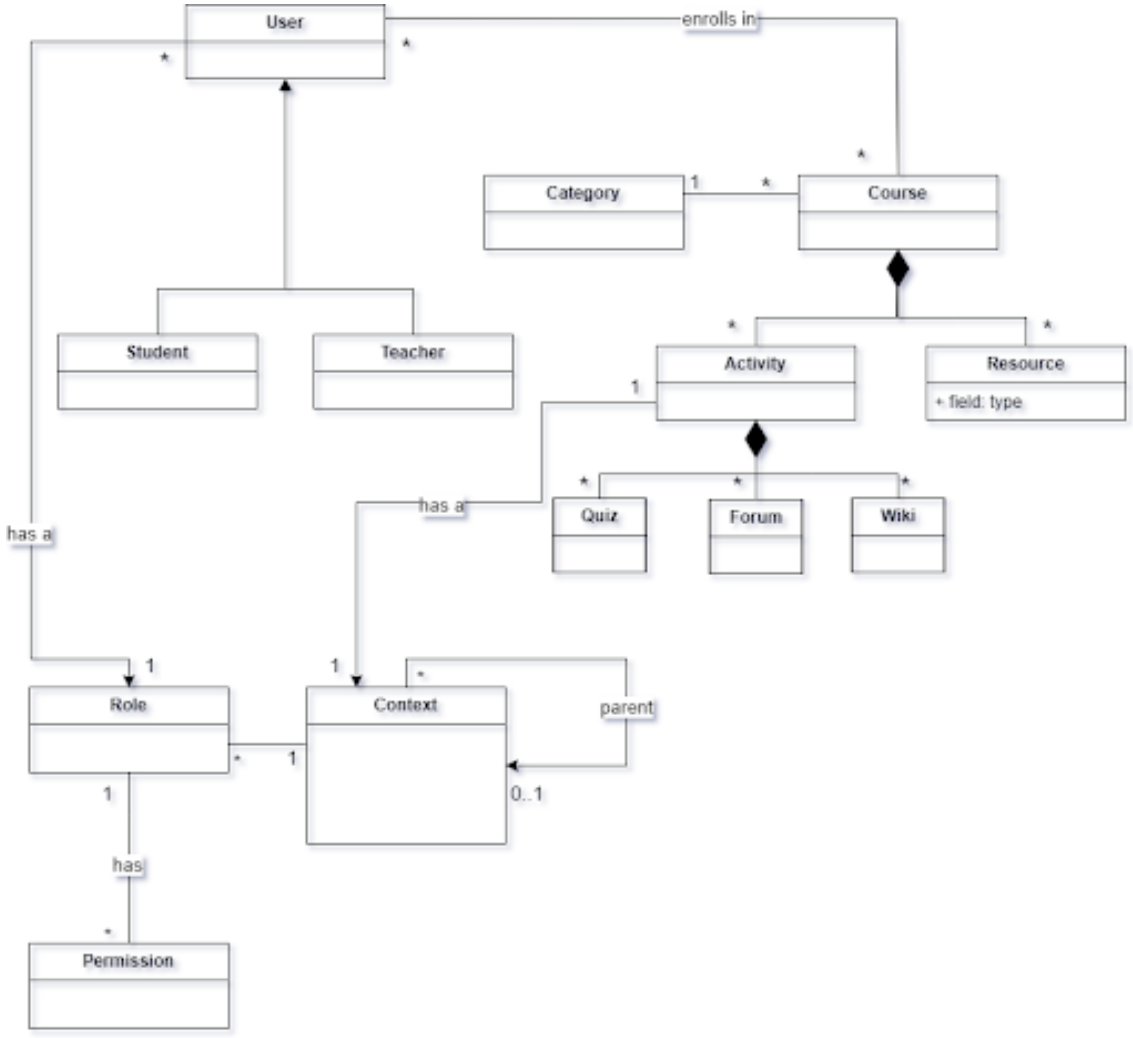

Fig. 1 – UML Diagram of Moodle's System

To understand this UML let's dive into its **entities** and their **relationships**:

- **User -** Users can be either students or teachers. Therefore, we created a **User** class with inheritance to represent both roles, as they have different functionalities within the platform.
- **Course -** Moodle sites are divided into courses, where several students can enroll. Courses can be aggregated into **categories,** which can be managed or created by teachers.

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Each course comprises several **Resources** and **Activities.** Since a resource can be an external file or link, like a PDF file, a page of HTML within Moodle, or a link to something elsewhere on the web, we decided to only add a field '***type'*** to represent them.

- **Activities** can be Moodle-coded resources, so we created a **composition** relationship to the classes **Forum, Quiz** and **Wiki.** When an activity ends, associated resources also conclude. Each activity is inside a context, that is a concept that we will explain later.

Another important aspect to note is that Moodle really needs to ensure that only authorized people have access to certain resources. To accomplish this, they introduced the concept of role, permission, and context. The author also talks about capabilities, but we don't think that this needs to be introduced in the UML, as they depend on the actual context, roles and permissions that a user has, and are not themselves an individual entity with features.

- **Context -** with this in mind, each context represents different areas within Moodle where users can have varying permissions. Contexts form a hierarchy like a file system, with the System context at the top level.
- **Role -** each role defines a set of permissions for users within a particular context. For instance, a user can have the *student* role in one course and *teacher* role in another. Roles are defined globally but can be redefined within specific contexts.
- **Permission -** represents the specific actions or functionalities that a user with a particular role can perform within a context. Permissions are assigned to roles and can be aggregated based on the roles assigned to a user in each context.

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Since we think that this authorization system is confusing, we also decided to draw a sequence diagram representing how the capabilities of a user for a specific resource is determined.

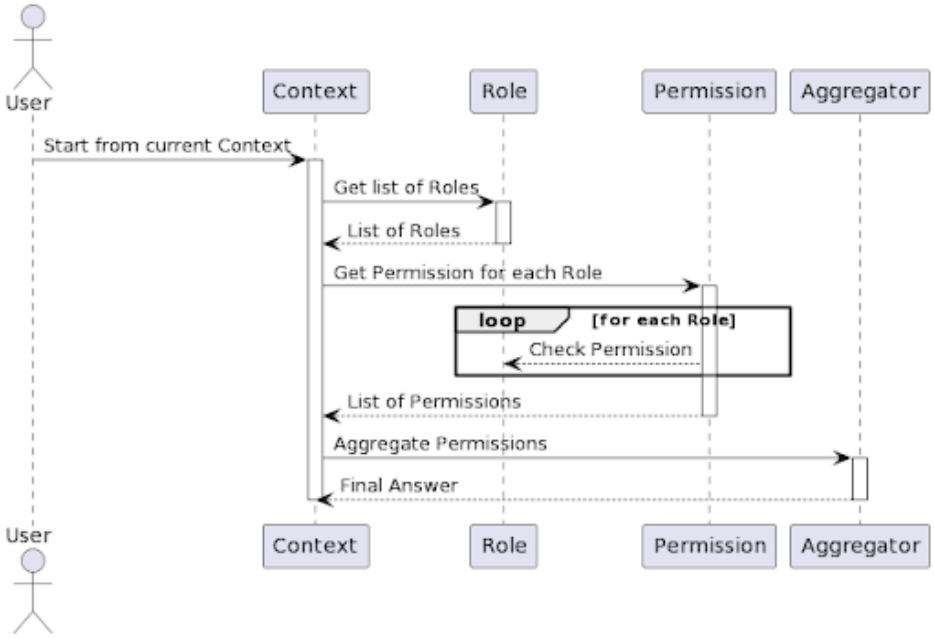

Fig. 2 – Sequence Diagram that represents the Context-Role-Permissions System

In this diagram the user acts as the initiator, triggering the process whenever the function ***'require_capability($context)'*** is called to access a resource.

The process begins by identifying the current context, that is passed as an argument in the function above, which could be an activity, a course or category context.

Next, the system retrieves a list of roles the user has within the identified context by querying the Role entity. For each role obtained in the previous step, the system retrieves the corresponding permissions. It's important to note that permissions represent specific actions or functionalities that a user with a particular role can perform within a context.

After obtaining permissions for all roles, the system aggregates these permissions to determine the users' final set of permissions within the context. This involves a process of combining and evaluating permissions from different roles to arrive at a conclusive decision, with the following logic:

- If any role gives the permission PROHIBIT for this capability, return false.
- Otherwise, if any role gives ALLOW for this capability, return true.
- Otherwise return false.

• Architecture Patterns

Moodle is built around a **plugin architecture** where the core system is surrounded by various plugins that extend its functionality. This approach allows for easy customization and

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

enhancement of Moodle in defined ways. Plugins in Moodle are strongly typed, meaning different types of functionalities require different types of plugins and APIs. This architecture pattern helps in modularizing the system, making it easier to manage and upgrade.

Moodle also implements the Presentation-Abstraction-Control (PAC) pattern. The **Presentation Layer** is responsible for the User Interface customization, where each page content is adapted considering the user role and its preferences; it also implements the Theme System. The **Abstraction Layer** takes care of Moodle's core functions like user enrollment and course management. It plays a crucial role in understanding how users interact with the system. By considering a user's role and permissions, the abstraction layer can process data in a way that's relevant to them. This ensures a smooth experience, providing users with the information and actions they need based on their specific context within Moodle. The **Control Layer** orchestrates the flow of control and communication between the Presentation and Abstraction components. It manages the interactions between the user interface elements and the core functionality based on the user's context. It ensures that the appropriate data is retrieved and processed according to the user's role and the specific context of the action.

Moodle follows a **Layered Architecture** where different components are organized into layers based on their functionality. For example, the core system, plugins, themes, and database operations are organized into separate layers, allowing for better organization, reusability, and maintainability of the system.

Moodle utilizes a **Database Abstraction Layer** that allows the system to interact with different types of databases without directly writing SQL queries. This layer abstracts the database operations, by defining the interface provided by the $DB global variable, making it easier to switch between different database systems without affecting the application's functionality.

Moodle also uses PHP global variables as an implementation of the **Thread-scoped Registry Pattern**. It plays a role in managing and storing thread-specific data or objects during the execution of concurrent processes. By utilizing this pattern, Moodle can ensure that data accessed and manipulated within a thread is contained and protected from interference by other threads, thereby enhancing thread safety and data integrity in a multi-threaded environment.

• System's Quality Attributes

Quality is an important part of software system design in general, as it has a significant impact on a product's success. In line with this, there are many quality criteria attributes, which serve the primary goal of assessing quality in various sectors. The most important quality criteria attributes, as described in Moodle's system architecture, are:

- **Usability**
- **Maintainability**
- **Interoperability**
- **Scalability**
- **Security**
- **Reusability**

Usability

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

This attribute refers to the user's experience with the end service/software, focusing on the effectiveness and overall user satisfaction. In this case, being an open-source platform, Moodle can adapt to each organization/user's needs, allowing many customizations and features to be implemented as "self-contained plugins". This personalization is also present in the way output code is generated, as it allows for different themes/skins to be used to change the appearance of the platform. Due to its internationalization, Moodle's user-friendly platform also considers the region of the user, adapting to its own language and specifications (like name and data formats).

Maintainability

Maintainability relates to the ease a developer is capable of fixing flaws in the system, without affecting other working components. To achieve this, it's crucial that the system follows software architecture rules/patterns and maintains consistency across the application. In Moodle's case, the division of the system into three main components - code, database and *moodledata* folder – has led to the simplification of updates and backups, allowing for the information to be available in multiple servers through load balancing.

Interoperability

Interoperability is the ability of two or more systems to communicate or share data. This includes operating systems, databases, and protocols. In this specific case, interoperability is achieved through the integration of different external systems, like authentication providers, student information systems or document repositories (like Alfresco), which guarantees that existing technological infrastructures of educational institutions can easily integrate with Moodle.

Scalability

This attribute refers to the system's ability to withstand increased demands without compromising performance. Moodle is perceived as a virtual learning environment and is capable of functioning both as an individual application or as part of a larger ecosystem within a school/university. Therefore, it can handle different sizes of user databases. Another feature to point out is the usage of plugins. Through its usage, new customizations and features can be implemented as self-contained plugins that interact with the Moodle core, through a defined API. This way, we're still able upgrade the core Moodle system, while adding new features to it.

Security

A system is secure if it is capable safeguarding data, applications and information from unauthorized entities. In the case of Moodle, its security is achieved through its Roles and Permissions system, where users, depending on the context, can have different roles and, consequently, permissions. In other words, a role is a set of permissions a user has in a specific context. For example, within a particular course, a user may be a Student and that role assignment will apply in the Course context and all the Module contexts within it. However, in another course, the user may have a different role. This is important for fine-grained control over the capabilities a user has in the system.

António Ferreira – up202004735
Francisco Serra – up202007723
Gonçalo Almeida – up202308629
João Maldonado – up202004244
Tomás Gomes – up202004393

Authentication is also important to highlight in this part. To restrict specific actions to only authorized users, the PHP code file for a plugin can have the line '***require_login'***, which verifies if a user is enrolled in or not, limiting the access to an activity or content.

Reusability

This attribute is the degree to which software components can be reused in other parts of the system. As described in the document, reusability in Moodle is ensured through its modular architecture and use of plugins. As it was mentioned before, by dividing the system into different components, plugins can be individually developed, shared and reused throughout different Moodle systems. It's also worth mentioning that the plugins in the plugin system of Moodle are strongly typed, meaning that each plugin not only has a specific API and follows certain conventions, but also, as mentioned, can be applied in every different Moodle installation.