T25 - Software Systems Architecture - M.EIC010

Rita Kiss - Laura Lumijärvi - Yana Peycheva - Juan José Osorio - Amanda Oliveira - Tiago Marques

Homework 6 - Understand Someone Else's Architecture

Part 1 - Assessment Rubric

Assessed team: T35

Diagrams	
Clarity	The separation in two main blocks, front-end and back-end, helps understand the main division between the user interface and the system itself in the diagram, and is clear due to them being labeled. One thing that is not clear at first sight is the different types of arrows that define the relations between components in the system, but once one looks at the label that is presented below the diagram, it becomes clear which type represents what behavior.
Consistency	Overall, the diagram seems consistent. One thing to consider in order to possibly improve the diagram is the spacing of some blocks, namely the ones inside the front-end main block and the ones inside the back-end main block. It's clear that the blocks that are near the edge of the corresponding main block, without having an arrow from a relation, are all consistent, with the same distance from its edge to the main block edge. Though, when they do have an arrow, the distance to the main block is different on the blocks on the front-end main block to those on the back-end main block. Also, possibly worth noting, still related to spacings, the two blocks that don't have a main block, the ones outside the front-end and the back-end, could be placed a bit lower, so that the parts of the relation arrows that are on the top and on the bottom would have the same length. It is perceptible that they are in the position they are to be in line, horizontally, with the other blocks from the front-end and the back-end. But maybe it would be better if they were positioned as was stated above: the two components would be centered, as the component "Reservations Manager" is centered and not in line with the others horizontally, and the label "Cooperative data exchange" would have more space to itself. To conclude, still in this area of the diagram, the label "Update Item's Status" could have been splitted in two lines, like the label above itself, "Cooperative data exchange". Note that these are only small details, but hopefully they can improve the consistency of the diagram.
Completeness	The diagram seems complete, there is nothing that would noticeably make it more readable. It already has a label that explains the fact that each box is a component of the system, the differences between each arrow type in a very succinct and clear way, and why the arrows point to where they point, which is the representation of the data's flow. The two main blocks, "Front-end" and "Back-end" do not need to be explained, this architecture is for people who are aware of these concepts and they are easily understood. So, no problem in leaving those out of the diagram label.

Sufficient Level of detail	The diagram contains a very good level of detail with lots of relations between the various components. It also presents the data flows for the main operations that the system will perform. The main division in front-end and back-end benefits the organization in the diagram, and the subdivisions inside each one and their relations increases the level of detail. Overall, it provides some extra and more deep information than what is considered to be the basics.
Text Description	
Clarity	The written part was clear, detailed, and efficiently conveyed the idea without unnecessary jargon. The descriptions were well-written in professional language and quite easy to read. To enhance clarity, I would suggest presenting some components of the diagram throughout the text again, making it easier to remember which part of the architecture the text is discussing.
Consistency	The text followed a logical order, consistently explaining the parts of the diagrams. It was well-structured with headings and subheadings.
Sufficient?	In addition to the small comments in the diagram and in the caption, the text provided detailed explanations and thorough inspection. It discussed both the diagram and the designed architecture comprehensively, leaving no questions unanswered. Moreover, considering the downsides and weaknesses of the architecture provided valuable insights and demonstrated that the team was aware of their design.
Correctness	
Anything missed?	After multiple analysis it was not clear that the architecture had anything missing. The diagram seems well detailed and the text describes each component in a clear way, providing further insight of the front-end and back-end parts of the system. In the document about the design of the system are also provided some additional considerations and even some usage scenarios, to make it more complete, comprehensive and plausible. In conclusion, if there is anything missing, it is not that relevant because the key components and ideas for the intended system are contained in the developed architecture.
Will it work?	The architecture looks good and seems to be well detailed. This architecture seems suitable for a library system as it addresses key functionalities such as catalog search and timely notifications. The clear diagrams and text descriptions enhance understanding. Overall, with further testing and refinement, this architecture has the potential to work effectively for the library system proposed.
Presentation	
Summary	Overall, we are confident that we could implement the design as it is described. Developing the whole system would require a lot more information, interaction with the client and hours developing and testing the system. But in terms of design and architecture, which is the focus of this assignment, it seems very plausible to be implemented. The architecture from group T35 was well thought and well transcribed in the text and diagram they presented.

T25 - Software Systems Architecture - M.EIC010

Rita Kiss - Laura Lumijärvi - Yana Peycheva - Juan José Osorio - Amanda Oliveira - Tiago Marques

Homework 6 - Understand Someone Else's Architecture

Part 2 - Moodle's Architecture

Introduction to Moodle's Architecture

Moodle is a learning management system providing a platform for online education and course management connecting students and teachers. Its most important quality attributes are scalability, flexibility, security, extensibility, and performance. Moodle's architecture adheres to modular and hierarchical design. At its core is a contextual hierarchy that organizes users' permissions and interactions. What this means is users navigate through various contexts like courses, categories, and activities, each with its own set of permissions and roles. This hierarchy mimics a folder structure, with the system context at the top level, followed by categories, courses, and activities. It allows for precise access control and permission management, ensuring that users have appropriate privileges based on their roles and context.

Additionally, Moodle employs a capability-based permission system, where each action corresponds to a specific capability. Administrators can define custom capabilities for plugins or modules, modifying access controls to meet specific needs. This mechanism ensures users only access authorized functionalities, enhancing system security and integrity.

Core Components

Moodle's architecture revolves around its core framework, plugin system, database layer, user interface, security features, integration points, and scalability mechanisms. These components work together to provide a well-structured and flexible learning platform.

The core framework consists of essential functionalities for user management, content creation, and course administration. The plugin system allows for customization and extensibility, enabling the integration of new features and activities. The database layer ensures seamless interaction with the database system. Moodle's user interface prioritizes usability and accessibility, ensuring a positive learning experience. Security features, including authentication and authorization mechanisms, protect user data and resources. Integration points

enable interoperability with external systems and services. Scalability mechanisms ensure optimal performance even when the amount of work/activity the system needs to handle varies. Combined, these components form the solid foundation of Moodle's architecture, supporting its adaptability and effectiveness as a learning management system.

Interaction Between Components

Various components interact seamlessly to deliver their functionality in Moodle's architecture. The core framework has a crucial role in initiating the system and managing interactions among plugins and other elements. As Moodle is opened, the core framework initializes essential services, such as the database abstraction layer and user interface components. Plugins extend Moodle's functionality and can interact with each other and the core framework through well-defined interfaces. The core framework facilitates communication between plugins, allowing them to utilize one another's features and resources. This modular approach ensures flexibility and scalability, enabling Moodle to adapt to diverse educational environments and requirements.

Customization and Extensibility

Moodle's plugin system authorize developers to customize and extend the platform by introducing new features and modules. This system enables the integration of tailored functionalities to meet specific educational needs and preferences. Developers can create plugins to add diverse functionalities, such as interactive activities, assessment tools, and communication channels. Utilizing the plugin system enables Moodle to be highly adaptable. Customization allows educators to tailor the learning environment to suit their pedagogical objectives and learners' requirements.

Security and Performance

Security features are crucial for protecting user data and system integrity. Role-based access control ensures that users can only access functions relevant to their roles, enhancing data confidentiality. Additionally, data encryption protects sensitive information from unauthorized access or tampering. Performance optimization techniques are also essential to ensure system responsiveness and scalability. These techniques include caching mechanisms and code optimization, which improve the platform's efficiency, especially on a large scale with a high volume of users/activities. Overall, Moodle prioritizes security and performance to provide users with a stable and efficient learning environment.

Scalability

Moodle's architecture is designed to scale horizontally and vertically, ensuring it can operate on growing user bases and increasing workloads. Horizontal scalability (scaling out) allows for the addition of more servers to distribute the load, while vertical scalability (scaling up) involves upgrading existing server resources to handle more simultaneous users. This flexible approach ensures that Moodle can adapt to changing demands without sacrificing performance or stability. By utilizing both horizontal and vertical scaling techniques, Moodle can effectively facilitate the needs of institutions of all sizes, from small schools to large universities.

In conclusion, Moodle's architecture is characterized by its modular design, hierarchical contextual model, and capability-based permission system. Moodle prioritizes scalability, flexibility, security, extensibility and great performance. The architecture enables flexibility, scalability, and access control while utilizing the plugin system, emphasizing security features, and optimizing performance. Moodle is a well-structured learning environment, and it enables customization and accommodating growing user bases through horizontal and vertical scaling.

Architecture patterns

Plugin Architecture

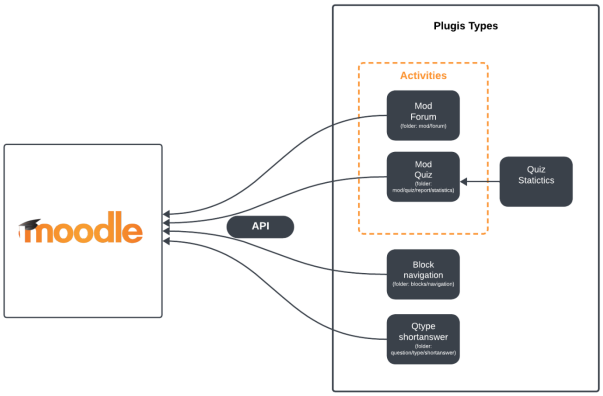


Figure 1. Plugins architecture

Moodle follows a plugin-based architecture where the core system is surrounded by various plugins that extend its functionality. This approach allows for easy customization of Moodle to meet specific needs without modifying the core system extensively. Plugins interact with the core through a defined API, enabling users to add new features or customize existing ones. The next example in Figure 1 shows that each activity module can have sub-plugin types. Right now, only activity modules can do this for two reasons. If all plugins could have sub-plugins, it might

slow things down. Activity modules are the main educational tools in Moodle, so they're the most important type of plugin and get special privileges.

Layered Architecture

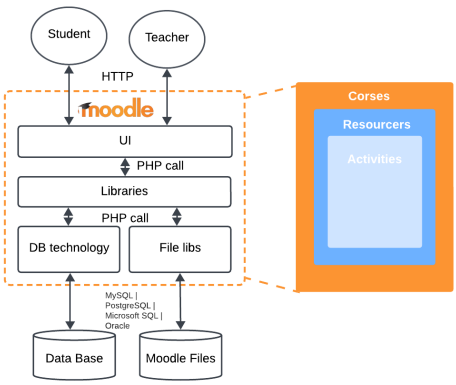


Figure 2. Layers architecture

Moodle's design incorporates a layered architecture where different components are organized into layers, each responsible for specific functionalities. For example, the system is divided into courses, which contain resources, activities, and users with different roles. This layered structure helps in managing and organizing the content and interactions within the system Also utilizes a database abstraction layer to interact with the underlying database system. This layer abstracts the database operations, allowing Moodle to support multiple database systems without directly coupling the application code to a specific database technology 1.

Model-View-Controller (MVC) Architecture

The platform needed to evolve from legacy scripts to a cleaner MVC architecture. This was necessary for to Moodle be able to improve code organization, maintainability, and scalability. This architectural pattern separates the application into three components: the Model (data and files), the View (UI, and the Controller (handles user input, using PHP).

Commented [1]: This was only was mention briefly on page 15.

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Homework 06

Understand Someone Else's Architecture

FEUP-ASSO- 23/24

Rita Kiss - Laura Lumijärvi - Yana Peycheva - Juan José Osorio - Amanda Oliveira - Tiago Marques

Software chosen:

Important attributes

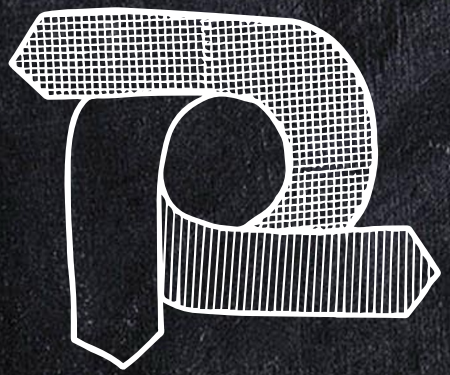
FLEXIBILITY

SECURITY

SCALABILITY

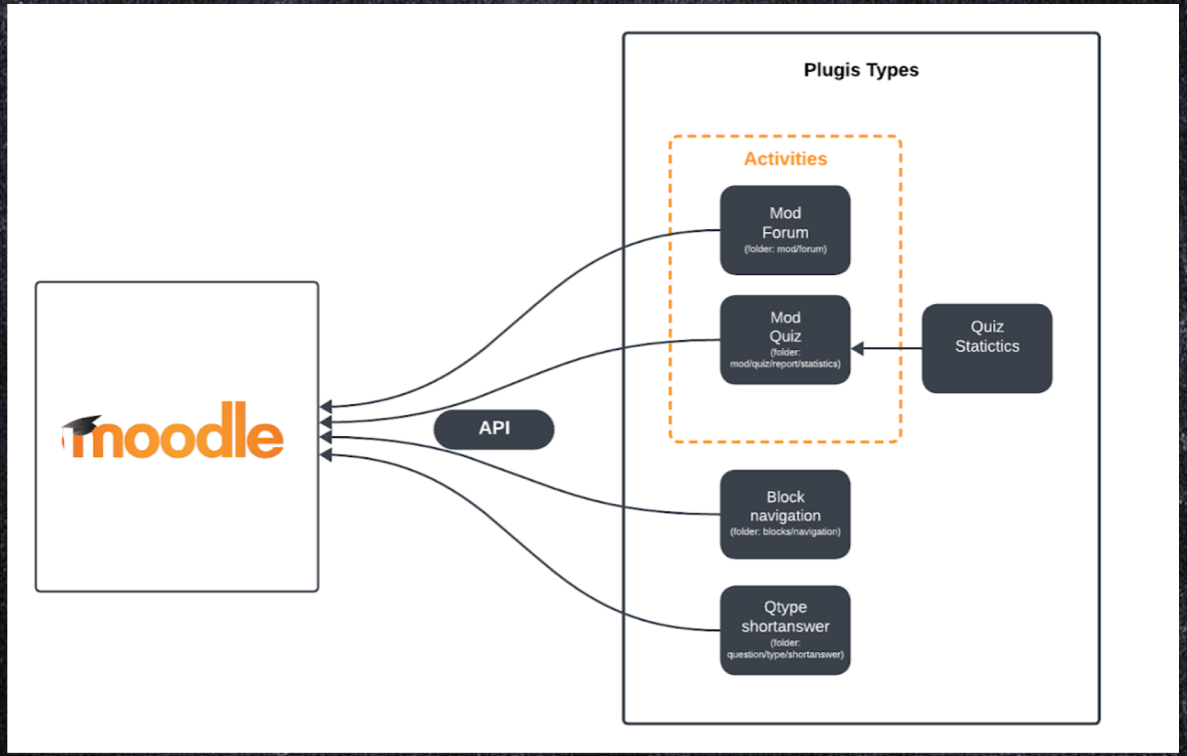
PERFORMANCE

Interaction Between Components



Various components interact seamlessly to deliver their functionality in Moodle's architecture. The core framework has a crucial role in initiating the system and managing interactions among plugins and other elements. As Moodle is opened, the core framework initializes essential services, such as the database abstraction layer and user interface components. Plugins extend Moodle's functionality and can interact with each other and the core framework through well-defined interfaces. The core framework facilitates communication between plugins, allowing them to utilize one another's features and resources. This modular approach ensures flexibility and scalability, enabling Moodle to adapt to diverse educational environments and requirements.

Plugins Architecture



U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Thanks!

Do you have any questions?